A REAL-TIME STRATEGY AGENT FRAMEWORK AND STRATEGY CLASSIFIER FOR COMPUTER GENERATED FORCES

THESIS

Lyall Jonathan Di Trapani, Captain, USAF

AFIT/GCS/ENG/12-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/12-04

A REAL-TIME STRATEGY AGENT FRAMEWORK AND STRATEGY CLASSIFIER FOR COMPUTER GENERATED FORCES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Insitute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science

Lyall Jonathan Di Trapani, M.Eng.E.E., B.S.E.E.

Captain, USAF

June 2012

AFIT/GCS/ENG/12-04

A REAL-TIME STRATEGY AGENT FRAMEWORK AND STRATEGY CLASSIFIER FOR COMPUTER GENERATED FORCES

Lyall Jonathan Di Trapani, M.Eng.E.E., B.S.E.E.
Captain, USAF

Approved:

// signed //	07 June 2012
Gary B. Lamont, PhD (Chairman)	Date
// signed //	07 June 2012
Douglas D. Hodson, PhD (Member)	Date
// signed //	07 June 2012
Maj Kennard R. Laviers, PhD (Member)	Date

Abstract

This research effort is concerned with the advancement of computer generated forces AI for Department of Defense (DoD) military training and education. The vision of this work is agents capable of perceiving and intelligently responding to opponent strategies in real-time. Our research goal is to lay the foundations for such an agent using the Balanced Annihilation real-time strategy (RTS) game running on the Spring engine. To achieve this goal, we design and implement an extensive RTS AI agent framework and a system to produce strategy classifiers. Six research objectives are defined: 1) Formulate a strategy definition schema effective in defining a range of RTS strategies. 2) Define eight balanced strategies using the strategy definition schema. 3) Design a real-time agent framework that, when given a strategy definition expressed via our strategy definition schema, plays the game according to the defined strategy. 4) Generate a quality RTS data set. 5) Create an accurate and fast executing strategy classifier. 6) Find the best counter-strategies for each strategy definition.

The agent framework is used to play the eight strategies against each other and generate a data set of game observations. To classify the data, we first perform feature reduction using principal component analysis or linear discriminant analysis. Two classifier techniques are employed, k-means clustering with k-nearest neighbor and support vector machine. The classifiers are tested using 5-fold cross-validation. The resulting classifier is 94.1% accurate with an average classification execution speed of 7.14 μs when running on a budget-class desktop personal computer.

Our innovative agent framework and strategy classifier have successfully laid the foundations for a dynamic strategy agent. Once the dynamic strategy agent concept is validated, these methodologies and techniques can be ported to computer generated forces of next-generation DoD wargames to enhance military training and education.

Dedicated to my lovely wife.

Acknowledgments

I would like to thank Dr. Lamont for his guidance, patience and encouragement throughout this research effort. It was an honor working under him.

Lyall Jonathan Di Trapani

Table of Contents

List of Figures

List of Tables

List of Abbreviations

A REAL-TIME STRATEGY AGENT FRAMEWORK AND STRATEGY CLASSIFIER FOR COMPUTER GENERATED FORCES

1 Introduction

Research in artificial intelligence (AI) for computer generated forces (CGF) in real-time strategy (RTS) games is of interest to both the U.S. military and academia. The following two sections motivate our research from the perspective of both establishments.

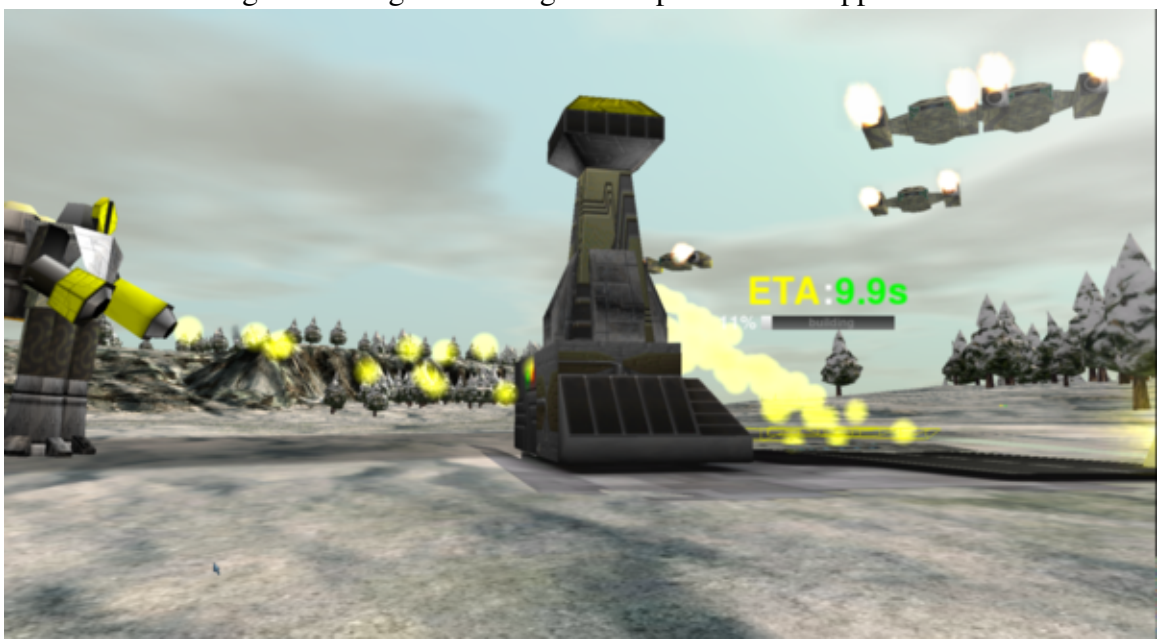
1.1 Military Wargames and RTS Games

Wargames, also termed military simulations [56], have been used to train and educate future strategic military leaders and decision makers for hundreds of years. As far back as the 600s, the Persian ruling class would play a chess-like, abstract, strategic wargame named *Chatrang* with their young princes to teach them to think about their opponent's choices and likely decisions—to teach them the critical thinking ability needed to make strategic decisions when commanding an army in war. Wargames have continued to evolve over the centuries from chess, a first generation war game, to second generation wargames, like risk, and into 3rd generation wargames, such as RTS games. Today, the U.S. military, like many other modern nations, use 3rd generation wargaming to educate and train their officers as well as attempt to predict the outcome of possible future battles. [11, 53]

1.2 Academic Interest in RTS Game AI

Recently, academic interest in RTS game AI has surged. Both the IEEE Conference on Computational Intelligence and Games (CIG) and the Association for the

Figure 1.1: Agent Sending Gunships to Attack Opponent



Advancement of AI (AAAI)'s Conference on AI and Interactive Digital Entertainment (AIIDE) have published several RTS AI related papers. [2, 19] Additionally, both conferences host yearly RTS AI competitions which have enjoyed an increasing number of submissions and attention over the last 3 years. [10, 24]

This interest in RTS AI started around the time M. Buro published a call for papers in RTS AI in 2004 and research effort in the field has remained strong ever since [8].

1.3 Research Goal

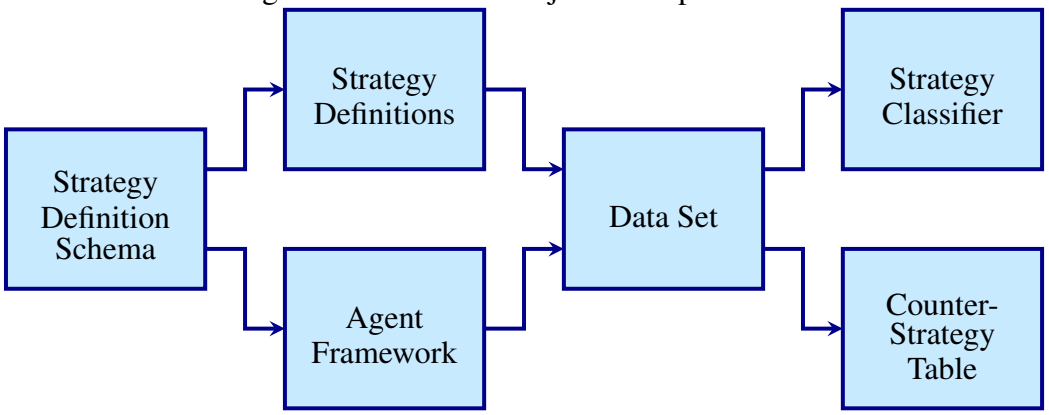
Our goal is to lay the foundations for a RTS agent development methodology that produces agents capable of making strategic decisions in real-time. We focus on a methodology appropriate for computer generated forces in Department of Defense (DoD) computer simulated wargames tailored towards military training and education. Our long-term vision is an agent able to perceive opponent strategies in real-time and intelligently respond to changes in opponent strategy.

To achieve this goal, we design and implement a RTS AI agent framework and a system to produce strategy classifiers. The framework and classifier produced must be appropriate for implementing an agent that can identify its opponent's strategy in real-time and adapt intelligently. By doing so, we provide the foundations of a practical RTS agent methodology which can be applied to CGF in DoD wargames.

1.4 Research Objectives

In order to meet our goal, we have defined 6 research objectives. Our specific research objectives are as follows.

Figure 1.2: Research Objective Dependencies



1. Formulate a strategy definition schema effective in defining a range of RTS strategies.

2. Define eight balanced strategies using our strategy definition schema.

3. Design a real-time agent framework that, when given a strategy definition expressed via our strategy definition schema, plays the game according to the defined strategy.

4. Generate a high-quality, strategy-focused, RTS game data set from game observations.

5. Create a classifier able to accurately, and quickly classify real-time game observations as strategies. The classifier must have an accuracy ≥ 0.85 and execute in $< 3ms$.

6. Find the best counter-strategies for each strategy definition on each map.

Figure 1.2 displays the dependencies between our 6 research objectives.

Figure 1.3: Heavily Defended Base Constructed by Agent Playing The Turtle Strategy

1.5 General Approach

We formulate a strategy definition schema consisting of a series of integer values that define how to play the game using a specific strategy. These integer values represent concepts, such as resource allocation into different categories (build-power, economy, defense, units), army composition, and how to construct an initial economy. Using our strategy definition schema, we define a list of basic RTS strategies, such as turtling, rushing, expansion, air power, etc. We create an AI agent framework that implements the

low-level tactics and takes as input a strategy definition, which it follows while playing the game. The agent framework is able to apply any strategy expressible in our strategy definition schema. We build the agent framework on top of the Spring engine [42] and design it to play the Balanced Annihilation (BA) game. Screenshots of the agent in action are presented in Figures 1.1, 1.3, and 1.4. We run the agent while playing different strategies over several iterations and collect game state observations (unit types, unit positions, economy statistics). Using the collected observations, we create various classifiers. They are trained using AI techniques to produce classifiers capable of classifying a given opponent game state observation into one of the pre-defined basic strategies. We create classifiers that learn the strategy centers via K-means clustering with K-nearest Neighbor (K-NN) as well as classifiers that learn the strategy boundaries via Support Vector Machine (SVM). We also perform two feature reduction techniques, Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), on the training set prior to training the classifier. Thus, we can compare the effect of applying PCA or LDA against using the raw data set without any feature reduction. Lastly, we find appropriate counter-strategies for each basic strategy by inspecting our data set of game observations.

1.6 Thesis Overview

The remaining of the document is split across 7 more chapters. Chapters 2 and 3 cover background topics. Chapter 2 gives a general background of RTS and AI concepts relating to the research. Chapter 3 provides an overview of previous RTS AI research approaches. Next, Chapters 4 and 5 present our methodology. Chapter 4 details our strategy definition schema, strategy definitions, and our agent framework. This chapter includes discussions on our agent architecture and design decisions. Chapter 5 lays out our data collection process, how we produce our classifiers, and the method we use to

Figure 1.4: Agent Constructing Base Defenses

discover counter strategies. Our design of experiments is presented in Chapter 6 followed by the results and discussion in Chapter 7. Concluding remarks and future work can be found in Chapter 8.

2 Background in RTS Games

2.1 Introduction

This chapter discusses background information pertinent to our research effort. We give a brief history on wargames, present an overview of RTS game concepts, define general RTS strategies, and discuss AI techniques used to meet our objectives. The background discussion is continued in the next chapter which explores previous approaches to AI research in RTS games.

2.2 Wargames

First generation wargames are abstract representations of war and were used to train and educate future leaders in strategic thinking. With second and third generation wargames, the fidelity of the games increased to allow more realistic simulations of war. Because of the increased fidelity, the use of wargames expanded from mere training and education into actual preparation and planning for real battles by using the wargames to predict the outcome of specific skirmishes. However, war has evolved to include strategic bombing and effects-based warfare with greater emphasis on command, control, communications, computers, and intelligence (C4I). As a tool for predicting the outcome of today's military engagements, wargames have not kept pace with the ever increasing complexity of modern war. [11]

For the past 70 years, our military wargames have consistently failed to predict the outcome of modern warfare battles. However, modern wargamming has grown in popularity in the civilian sector. Previously, second generation wargames were hand-crafted by individuals with the correct mechanical, artistic, and technical knowledge and an interest in wargrames. Thus the supply of wargames was limited. Improved manufacturing after the industrial revolution drastically increased the supply and reduced

the cost; making second generation wargames available to the middle class. The advent of the affordable personal computer has further increased the accessibility and ease of use of wargames while simultaneously decreasing the cost of ownership and operation; thus, spawning a multi-billion dollar RTS computer game industry with millions of players world-wide [11]. Today, third generation wargames are readily available on multiple computing platforms—most at reasonable prices and some completely free.

2.3 RTS

The first RTS game was a Herzog Zwei published in 1989 for the Sega Mega Drive/Genesis gaming system. However, the genre did not attain critical acclaim until Dune II was released on DOS in 1992. Several other seminal titles appeared in the 1990's which firmly established the genre, such as the Warcraft series, the Command and Conquer series, Total Annihilation, Age of Empires, and Starcraft. Today, RTS games are a staple in PC video gaming. A more comprehensive RTS game history is available in [54].

RTS games provide a compelling platform for military wargames for the purpose of training and education. Current RTS games, although quite detailed and complex for an entertainment-focused game, are still just abstract models of real, modern warfare and as such, are not suitable to be used as "battle outcome prediction tools." However, they do enable and encourage the players to think through and implement complex strategies in addition to dynamically adapting or changing their strategy to the fluidity of the battlefield. In this way, RTS computer wargames have considerable potential in teaching players to think strategically on the battlefield and think critically about his or her opponent's thoughts and potential actions. These are crucial skills for successful military officers.

Due to the growth of the civilian RTS wargamming market, civilian RTS wargames have two obvious advantages over military developed wargames. The first is the cost. In the past, the U.S. military has spent many millions and in some cases, billions of dollars

on creating their own wargames [11]. Today, a civilian commercial RTS wargame can be purchased for < $70.

The second benefit to using civilian RTS wargames is accessibility. RTS wargames are available on many computing platforms, such as Windows, Linux, FreeBSD, and MacOS based PCs as well as home consoles such as the XBox 360 and Playstation 3.

Furthermore, they do not require a group of players to meet together along with a moderator to arbitrate the game. RTS games can be played in single player mode using built-in RTS AI agents. The agent takes the place of a human opponent. It performs the same basic tasks of a human player. These tasks often include building a base, constructing units, and attacking the player. The purpose of commercial RTS AI is to provide the human player a challenging, entertaining opponent for off-line play or simple practice. Unfortunately, many commercial RTS strategy games have very poor AI. Developers choose not to implement modern AI techniques for various reasons. Some of the reasons being:

- Perception that modern AI algorithms are too slow to implement in a real-time program.
- Developers do not know about the existence of AI techniques.
- Developers do not understand how to implement AI techniques.
- Harbored distrust of modern AI techniques over traditional simple video game AI such as state machines and scripted actions.

Furthermore, AI in RTS games is usually a low priority for the developers. Most of the simulation logic must be implemented before the AI agents can be fully tested. Additionally, commercial gamer attention is focused on the multi-player game-play and graphics, therefore, so is the money [20]. Because AI is a low priority, commercial RTS

AI developers most often employ a scripted AI or include cheating, such as perfect information or resource bonuses, in their agents.

There is a clear need for further research in applying modern AI techniques to the domain of RTS games [8].

2.4 The Levels of RTS AI

Three levels of RTS AI are discussed in order to provide an understanding of possible design structures that can implement the desired AI features. They include Unit-level AI, Tactical-level AI, and Strategic-level AI.

2.4.1 Unit-level AI. This is the lowest level. It deals with controlling individual units directly. It is used whenever a decision can be made without consideration for other friendly units. Actions such as moving a unit, firing on a target, positioning a unit in its optimal firing range, or using a unit to perform scouting are examples of unit-level AI. This is also referred to as the *"micro"* level of RTS gameplay.

2.4.2 Tactical-level AI. This level is concerned with making decisions at the group or squad level. Actions such as choosing what direction for a group to attack the enemy base from, selecting a target for a group to attack, and arranging a squad formation are examples of tactical-level AI. This level is usually focused on combat.

2.4.3 Strategic-level AI. At the strategic level, the AI is concerned with the overall approach to winning. This is where the big decisions are made. Strategic-level AI mainly deals with three categories: economy, construction, and timing. Examples of strategic decisions include how much resources to spend on building economy vs offensive units, what unit types to build, and where to erect defensive structures. This level AI also includes timing decisions such as when to increase one's technology level, when to push for a large attack on the enemy's base, and when to expand one's base to control more

resources. This level AI is often referred to as the *"macro"* level of gameplay. Regarding strategic-level AI features, three important decisions are defined.

1. *Teching*: This refers to the time chosen to advance to the next technology level and what technology branch to pursue. In order to advance to the next technology level, the player must first develop its economy to a point which it sustains the requirements of the next technology level. The production of new factories or construction units typically causes a drop in military offensive and defensive unit production as resources are redirected from offense/defense to the new factories and/or construction units. During this time, the player is more vulnerable to attack, and thus, choosing to tech right before the opponent attacks can be disastrous. So we see teching is a double-edged sword. Teching too early strains the economy, causing production to drop. Teching too late gives the opponent the advantage—the opponent will advance to the next technology level and produce stronger units that will crush the lower technology units of the player. However, teching at the right time gives the player the technology advantage which is often a major factor in determining who will be the victor. Equally important to choosing when to tech, is what technology branch to pursue. For example, the player can choose to advance its technology level in land units, sea units, or air units. Choosing the right branch for the given map and opponent's play style is often the difference between winning and loosing.

2. *Pushing*: Pushing describes the time chosen to launch attacks. Launching an attack with an army of insufficient size or incorrect unit types just waste the army as it is destroyed by the opponent's defenses. Delaying in launching an attack with an army of correct size and composition is also wasteful, as this gives the opponent time to erect stronger defenses, fortifications and construct a better army.

3. *Expanding*: This describes the time and place chosen to create the next base expansion. Expanding to soon can put the player at a disadvantage as it is forced to assign construction units to the new expansion and defend the expansion, thus putting strain on the economy. If the player is not ready to defend the new expansion, its opponent may destroy the expansion at will, causing the player to loose its valuable construction units and new buildings. If the player expands too late, it is also at a disadvantage because its opponent will have already expanded and be reaping the benefits of the stronger economy, and thus, create a more powerful army and obliterate the player who delayed in expanding.

2.5 Definition of RTS Terms

The following list defines RTS terms we use throughout the remainder of the document. Such definitions reflect those used by the RTS AI development community, and particularly, the Spring Engine development community.

1. *Real-time*: Gomaa distinguishes between hard real-time applications and soft real-time applications in that *"a hard real-time system has time-critical deadlines that must be met to prevent a catastrophic system failure. In a soft real-time system, missing deadlines occasionally is considered undesirable but not catastrophic."* [16] According to this definition, RTS games, and as an extension, RTS game agents are soft real-time applications. The RTS agent's *"deadline"* is to complete all processing for the current frame within a game cycle period. This is misleading, however, because the agent does not own all of the compute resources to itself. The resources are shared with the rest of the game's simulation. Therefore, the entire game simulation, including the agent, must complete processing each frame within a game cycle period. Missing this deadline does not cause a catastrophic system failure, but merely gives the user a potentially unpleasant experience as the game

simulation rate slows down to accommodate the extra computation. Because the Spring engine runs on Linux with a non-real-time kernel, it is difficult to guarantee absolute execution bounds. For this reason, we use first and second order statistics to show that our average execution times complete well within the deadlines. This is sufficient to meet the soft real-time requirement.

2. *Map*: A map is an instance of the virtual battlefield used to conduct a game. It is the data structure that defines the environment, such as terrain elevation, terrain type, terrain features, color, etc. The characteristics of the map influences the players' decisions and impacts the effectiveness of certain strategies. For example, a map 80% covered by water may influence the players to use sea or air power instead of relying on ground units for offense and defense. We use three different maps in our agent framework testing.

3. *Economy*: This refers to the production of resources required to create units and buildings. In BA, the resources are metal and energy which are produced using metal extractors and solar panels respectively.

4. *Worker time*: Each construction unit and factory has a *worker time* attribute. Its value determines how fast a factory or construction unit can build new units and buildings.

5. *Build power*: This is the players total capacity to construct units and buildings. It is computed as the sum of the factories' and construction units' *worker times*.

6. *Build time*: Every unit and building has a *build time* attribute. It relates to the unit's cost and represents the specific amount of accumulated build power needed to fully construct the unit.

2.6 RTS AI Agent Problem Solving

RTS games have several interesting problems for an agent to solve, many of which have large, rugged search spaces. [8]

Economy and Resource Management. An agent must make decision about what units and buildings to construct. It must also plan a base layout and carefully select its building locations to give it a strategic advantage. The construction of resource-collecting structures, such as solar panels or metal extractors must be planned ahead of time so that the appropriate resources are available when needed.

Combat. The player must assign units to attack groups. If there are n units and m attack groups, then the player has $O(m^n)$ choices. Also, when an attack group engages an enemy's attack group, the player must assign units to attack opponent units. This is also many-to-many matching problem with exponential choices.

Path finding. Whenever a unit is moved, a path must be computed. This computation must take into account static obstacles (buildings, mountains, crevices, rivers, etc.) and dynamic obstacles (other units). Optimally solving a single instance of this problem without any time constraint is not difficult. However, an RTS game can have several hundred units in play. Solving hundreds of dynamic path-finding problems in real-time becomes a difficult problem. Furthermore, just avoiding obstacles in not enough. An agent should also generate path trajectories that avoid unwanted enemy confrontations while placing the agent's attack group in an optimal attack position to strike its target.

Non-deterministic and real-time. To be competent at playing RTS games, an agent must be able to cope with the constantly changing environment of the game. Just because it choses to take a specific action does not mean the action will be successful. For example, an agent sends a construction unit to build a metal extractor, however, the construction unit is destroyed before it reaches the build site.

Imperfect/partial Information. We have given examples of problems the agent deals with in RTS games. However, these problems are all handled under partial information. The agent can only see what its units see. The agent does not know what lies ahead unless it has a unit scouting the area.

2.7 General RTS Strategies

There are a number of "common" RTS AI agent strategies including Rush, Blitz, Turtle, Expansion, and Super-Weapon. These five presented strategy descriptions are based on discussion found in [44].

2.7.1 Rush. In a rush strategy, the player focuses resources on building a small, initial, offensive, military force with which to attack its opponent as early as possible. Thus, the player makes a deliberate decision to dedicate most of its construction units and raw materials to building an army as opposed to expanding its economy. The player only develops its economy enough to establish an initial military production facility and produce a small number of units. The intent is the initial assault arrives before the opponent can create any meaningful defense and thus cause a great deal of damage to the opponent's economy and military production. The player can then follow up with similar small waves of attacks to finish off the opponent.

This strategy works best on small maps or in games where the players' starting locations are in close proximity to one another. This strategy is less effective on large maps where the players' starting positions are far apart and the initial attack force must travel a considerable distance before reaching the opponent's base. This time delay gives the opponent the necessary time to build up sufficient defenses to ward off the attack, preventing the player from significantly damaging its opponent's economy or military production.

The opponent's decisions in the opening of the game are crucial to the success or failure of a rush strategy. If the opponent does not expect a rush attack, he may chose to focus his resources in developing his economy in an attempt to build a stronger economy than the rushing player. If so, the initial attack on the opponent's base will be devastating. On the other hand, if the opponent prepares for a rush attack by building some cheap initial defenses, such as gun emplacements or assault units, the rush attack will inflict minimal damage and fail to gain any economic advantage over the opponent. This may cause the opponent to be economically ahead of the rushing player and eventually cause the rushing player to lose the game.

Rush strategies can be implemented using different units. For example, an infantry rush uses weak, low-level, raider units while a tank rush uses basic tank units. The same principles apply to any implementation with some adjustments. For example, a tank rush requires a stronger economy than an infantry rush.

2.7.2 Blitz. A blitz strategy is primarily an offensive strategy, although not as extreme as a rush strategy. A blitzing player constructs an army of moderate size—strong enough to drive straight into the enemy base, overcome any defenses, and destroy the base in one attack. A blitz is appropriate when the player expects its opponent to prepare for a rush. By creating a larger army than with rushing, the player is able to prevail against the small defenses the opponent established, and win the game. Compared to rushing, a blitz attack requires a stronger economy as it constructs a larger army.

2.7.3 Turtle. A turtle strategy, also called *porcing*, relies on defense to bring the victory. The player constructs defensive structures such as laser turrets and walls while building-up its economy. Since defensive structures are usually more cost-effective than offensive units, the player may be able to pull ahead of its opponent economically. Later in the game, once the player's economy is strong, it shifts its focus to producing a

powerful army which the player uses to crush its opponent in one attack or the player builds a super-weapon and takes its opponent out from a distance (see super-weapon strategy below.)

Turtle strategies can be difficult to maintain as the game progresses. As the base increases in size and the player expands into larger areas, its larger territory becomes more difficult and expensive to defend. While the player is forced to defend all of its territory well, its opponent only needs to punch a small hole in the player's defenses to sneak an attack into the base.

Turtling is often used by novice players as it allows them to focus only on their own territory and ignore their opponent. Strategies like rush and blitz require the player to multi-task: controlling both the construction units in its base and the attacking units assaulting the enemy's base.

2.7.4 Expansion. Expansion, also known as map control, is a strategy where the player focuses on controlling as much resources as possible. The player is always trying to expand its base over uncontested resources to gain an economic advantage. This economic advantage allows the player to outproduce its opponent in military strength, eventually leading to victory. An expansion player must keep its opponent occupied by harassing the enemy base with light raiding parties. This harassment tactic, if successful, will keep the enemy from noticing undefended expansions long enough for the player to send defensive units or erect defenses. Expansion strategies can be difficult to execute as the player must manage multiple satellite bases—coordinating both new constructions and defenses at all sites, in addition to orchestrating occasional raids against the enemy.

2.7.5 Super-weapon. With a super-weapon strategy, the player focuses on defense while building up its economy. Later in the game, when its economy is strong, the player

constructs a nuclear missile silo or a very long range cannon to destroy the enemy from the safety of its own base.

2.8 AI

AI is concerned with the study and design of rational agents [34]. Most of AI can be categorized into two main disciplines; search and knowledge representation.

Famous successes in AI research include the chess competition between IBM's Deep Blue computer and Garry Kasparov, the then reigning world champion of chess. Deep blue failed to win the first competition in 1996. However, in 1997, Deep Blue defeated Kasparov during their rematch, winning 3.5 of 6 games. IBM researchers used a brute force mini-max algorithm with alpha-beta pruning to search approximately six to eight moves ahead in an attempt to find the best action to take. The mini-max evaluation function had many parameters which were optimized by analyzing thousands of master chess games. IBM employees, with the help of chess grand masters, also created a hand-coded opening book with 4,000 positions and 700,000 grand master games in addition to an endgame database of moves fine-tuned to Kasparov's chess techniques. [52]

Another, more recent, famous, example of AI success, also from IBM, is Watson, the *Jeopardy!* playing computer. Watson defeated two *Jeopardy!* champions. The computer was pre-loaded with several databases of trivia information, such as the entire wikipedia encyclopedia text, totalling four terabytes in size. It used multiple database search algorithms and natural language processing algorithms to successfully answer more game-show questions correctly than its opponents. [55]

2.8.1 Weak AI in RTS Agents.

Strong AI is the field of AI that attempts to match or exceed the intelligence of humans. Weak AI is the opposite; it does not attempt to match or exceed the intelligence of humans. This is appropriate for video game AI, as the AI is only intended to entertain the human opponent by providing the illusion of true

intelligence. RTS AI agents in commercial and open source video games often apply simple, computationally-efficient AI approaches such as scripted behavior triggered by if-then-else rules, state machines, and greedy search algorithms. We apply a factored state machine agent with greedy search to implement our agent framework. [34]

2.8.2 Statistical Machine Learning. Statistical machine learning is largely involved in solving three problems; clustering, classification, and regression. Techniques in machine learning can be categorized as supervised and unsupervised. A supervised technique involves using labeled samples to train a model. Supervised techniques include K-NN, LDA and SVM. An unsupervised technique, on the other hand, does not use labels to train the model. Examples of unsupervised techniques are k-means clustering and PCA. All the above techniques are employed in creating and testing our strategy classifiers. We apply k-means, a clustering algorithm, together with K-NN to construct classifiers. We also use SVM [13] to construct classifiers. PCA and LDA are applied for feature reduction, also known as dimensionality reduction. K-fold cross-validation is a popular method to perform testing on classifiers as it achieves an acceptable balance between bias and variance. We use 5-fold cross-validation in our testing. First and second order statistics are used to measure accuracy and precision respectively. For more in-depth discussion on statistical machine learning, see [14, 17].

2.9 Chapter Summary

In this chapter, we provided background information on our research effort. This included a brief history on wargames, an overview of RTS game concepts, the definitions of general RTS strategies, and an overview of AI techniques used to meet our objectives. The AI section is not intended to discuss the complete field of AI [34, 14, 17], but only to introduce AI aspects of use in an RTS agent.

Using the Chapter 2 foundation discussion, the next chapter presents previous approaches of research in RTS AI.

3 Previous AI Approaches in RTS

3.1 Introduction

Interactive computer games have been an attractive platform for AI research for over 12 years due to their low cost and accessibility [21, 20]. Over the years, RTS games have become especially popular as research platforms because of their complexity and many subproblems with large, rugged search spaces [8].

In the following sections we discuss a number of prevalent techniques used in recent RTS AI research. Each section is devoted to a single technique. The methods discussed are reinforcement learning, dynamic scripting, Monte-Carlo planning, case-based planning, Bayesian networks, evolutionary algorithms, modular agent design, and opponent modeling. This layout is inspired by [49].

3.2 Reinforcement Learning

Reinforcement learning is a machine learning technique where actions of a given state are rewarded or punished according to the fitness of the result they produce. The agent picks actions with the greatest benefit in an attempt to maximize some reward [38].

The contribution of [3] is an evaluation function suitable to predict the outcome of Spring BA games during different phases of the game. The function was defined with a number of parameters which were tuned using temporal difference learning over hundreds of games.

The authors of [45] present a method to create dynamic formations in the ORTS RTS game [9]. They use a set of parameters to define a formation. The "good" formations are learned via off-line reinforcement learning [38] with stochastic optimisation. Each opponent is played iteratively until a maximum number of iterations is reached (200 to 500). The learning weights of the parameters are updated after every game. Thus one

formation is generated for each opponent. The authors employ off-line opponent modeling to learn features of specific opponents. During gameplay, the opponent is classified using Bayes' theorem according to its observed features and a formation is selected based on the predicted opponent model.

The learning is entirely off-line. This approach requires the AI designer to be able to train the agent on all the opponents a priori.

3.3 Dynamic Scripting

Dynamic scripting is a form of online reinforcement learning [38] for agents with stochastic optimisation. It was pioneered by the authors of [37]. To apply dynamic scripting, one must have a deep understanding of the problem domain and use that knowledge to manually create a large number of tactics, also termed rules, as well as design an accurate evaluation function. The tactics are placed into a database and each assigned an equal weight. During gameplay, a tactic is probabilistically selected based on weight. The agent implements the tactic and the evaluation function determines the tactic's performance. If it does well, it's weight is increased, and all other tactic's weights are decreased. If it does poorly, it's weight is decreased, while all others are increased. This method was tested against static scripts, random scripts, and pseudo-random scripts (the opponent uses the same tactic as long as it is successful. If the tactic fails, the opponent randomly selects a new tactic.)

3.4 Monte-Carlo planning

With Monte-Carlo planning, one randomly simulates many plans, and picks the plan with the best performance. Plans are often constructed using a tree structure. Defining a proper evaluation function is very important, as an inaccurate evaluation function could lead to choosing bad plans. Thus, one needs enough domain knowledge to write a good

evaluation function. Also, Monte-Carlo planning requires one to have the domain knowledge to define the high level actions that make up a plan.

The approach requires a simulator to simulate plans. With RTS games, the game itself can serve as the simulator. This results in a very accurate, yet computationally expensive simulator. A drawback to Monte-Carlo planning is it does not learn from past mistakes. Monte-Carlo planning is applied on the ORTS game using the capture the flag gameplay mode in [12]. In [6], a Monte-Carlo planning algorithm is applied at the tactical assault level in the RTS game Wargus. The algorithm is concerned with choosing a group's defense, offense, group movement and positioning in tactical engagements.

3.5 Case-based Planning and Reasoning

With case-based plaining, one considers the current "case", or game state, and searches through a case-base (database of cases) to find similar cases. Of the similar cases, the action leading to the highest fitness value is selected.

To create the case-base, randomly pick a plan and execute it. If the evaluation function reports a positive fitness, increase the score for that plan/state pair. If instead, the fitness is negative, decrease score for that plan/state pair. Another way is to create the case-base by learning plans through observing expert players. This requires expert goal/action annotations.

In [27], the author attempts to achieve the goal of winning through a plan of subgoals. The author had 35, 23 and 17 recorded games from player A, B and C respectively. Each of the three sets of recorded games became a case-base. When the agent used the case-base from A, it played like A. When using the case-base from B, it played like B—likewise for C.

A case-based reasoning approach is used in [4] to create a case-based adaptive game AI which can rapidly adapt to its opponent and environment without requiring numerous trials to learn new behavior. The approach uses a case-base and game observations with

reinforcement learning to adapt AI behavior similar to the approach with dynamic scripting. The evaluation function is taken from previous work by the same author [3].

In [48, 47], the author uses case-based reasoning to decide build order in an imperfect information environment using the Starcraft RTS game.

The authors of [1] use case-based plan selection to adapt the work of [29] to be suitable against dynamic opponents instead of only statically scripted opponents.

3.6 Bayesian Networks

The author of [40] uses a Bayesian model to predict the opponent's tech tree. The models are trained using a database of over 8000 Starcraft game replays. In [41], the author leverages hierarchical Bayesian networks to perform local unit control. The author again leverages Bayesian models to predict the opponent's opening moves in [39].

3.7 Evolutionary Algorithm

The authors of [29, 28, 30] improve their adaptive AI by building on the concepts of dynamic scripting. To create the database of tactics, they use an off-line evolutionary algorithm to evolve tactics automatically instead of the previous way of hand-coding tactics. They name their system ESTG for Evolutionary State-based Tactics Generator. The game state is defined by the types of buildings the agent possesses. Whenever a building of a new type is constructed, the game transitions into a new state. There are 20 distinct game states for their testing platform, Wargus. The off-line evolutionary algorithm learns good tactics for each game state. The on-line reinforcement learning algorithm associates weights with the tactics/state pairs. As with dynamic scripting, the weights determine the probability of a tactic being selected in a given game state. This approach only considers the agent's own state and the previous performance of tactics in that state to decide the next action; it ignores the opponent's state entirely. The approach was tested successfully against four hand-written, static scripts.

3.8 Modular, Integrated Agent

In [25], the authors use human analysis of expert human play to define threshold values for simple, discrete decision-making. Their research platform is their ABL reactive planner on top of the open-source RTS, Wargus. Their architectural approach breaks the agent up into several components termed managers. The managers are strategy (initial and tier), production, income, tactics, and recon. They do not use special formations or low level tactics. The approach was tested against two static scripts: soldier rush and knight rush.

3.9 Opponent Modeling

The author of [35] applies hierarchically-structured opponent modeling to perform real-time strategy classification using the Spring engine with BA. The hierarchical structure consists of two levels: a top level used to classify general play style (aggressive or defensive) and a bottom level used to classify the specific strategy. For example, the aggressive play style can use a tank or k-bot strategy while the defensive play style can use the super weapon, tech, or bunker strategies.

The top level classifier uses a fuzzy model to distinguish between aggressive and defensive. The percentage of time the opponent is attacking indicates how aggressive it is. The classifier detects aggressiveness when a player unit is destroyed. The bottom level classifier uses observations events (under imperfect information) to classify opponent strategy. This occurs when attacking our scouting. The classifier emphasise recent events via discounted reward.

In [5], the authors incorporate opponent modeling to enhance the performance of their case-based adaptive game AI. They create the opponent models off-line. To create their opponent models, they use their domain knowledge of the game to manually select features used to define the models. This, of course, limits the accuracy of the models. The

opponent they use is the Spring AI agent named “AAI”. They play 975 games on three distinct maps—325 per map. For each game they collect feature observations of the opponent every few seconds via perfect information. K-means clustering is applied to the observations to generate opponent models. The models produced are actually opponent preferences of the “AAI” agent when playing on the three different maps.

To detect the opponent’s model, they use on-line classification via minimum euclidean distance from current game observation to cluster centers. Again, perfect information is used to collect the on-line game observation. It took about 10 minutes of game play to establish models.

The author of [51, 50] uses the spring engine to create a classifier that can predict game winning and loosing states in the third quarter of the game with an average accuracy between 65.8% and 76.6% depending on the opponent the classifier was trained on. The author creates four scripted opponents and four matching classifiers. Each of the four classifiers are trained using games where the matching opponent is player two. A dynamic agent is implemented that can win against each of the four static, scripted opponents when using the appropriate classifier trained on the target opponent.

The research effort of [23] is similar to the work presented in this thesis, except it is applied to the domain of football. Instead of classifying game state as strategies, they classify football formations as football plays. More specifically, the authors present an approach to online opponent modeling to improve offensive performance in the Rush 2008 football simulator. SVM is used as the classifier. Once the opponent’s football play is predicted, the agent selects an appropriate counter-play that will improve the yardage gain and executes it. The authors extend their work in [22] to include real-time plan repair using Upper Confidence Bounds applied to Trees.

3.10 Chapter Summary

The above research provides several implementations for tactical and unit-level AI. This includes individual unit control, dynamic formations, optimised tactic selection and group-level tactical assault control. These contributions could be integrated into a single agent capable of playing well at the unit-level and tactical-level.

At the strategic-level, contributions included build-order prediction, tech-tree prediction, and an RTS evaluation function. Our work in the strategic-level could be enhanced with the other strategic-level contributions, and eventually integrated together with the tactical-level and unit-level AI to complete the hypothetical agent.

In this chapter, a number of prevalent techniques used in contemporary RTS AI research are detailed. The methods discussed are reinforcement learning, dynamic scripting, Monte-Carlo planning, case-based planning, Bayesian networks, evolutionary algorithms, modular agent design, and opponent modeling. In the next chapter, an RTS AI agent framework is developed based in-part on the foundations discussed in Chapters 2 and 3.

4 Agent Framework

4.1 Introduction

This chapter is the first of two chapters concerned with our methodology. Here the focus is on the agent framework while the next chapter focuses on classifying strategies. In the following sections we present the strategy definition schema, formulate eight strategy definitions, and explain the design and implementation of our agent framework.

4.2 Strategic Constructs

We define a list of basic RTS strategies, such as turtling, rushing, expansion, strategic bombing, etc. The strategy definition schema is presented; it consists of a set of integer values (build-power, economy, defense, units, group composition and initial economy). All strategies are defined using this simple schema. The AI agent framework implements the low-level tactics and takes as input a strategy schema. The agent is thus able to follow any strategy expressible in our strategy definition schema.

4.2.1 Strategy Definition schema. A strategy definition schema is needed to represent and describe various strategies in a concise and flexible manner. The schema we develop only uses $6 + n$ integer variables where n is the number of combat unit types available. The strategy definition schema is shown in Table 4.1. The first four variables determine how the current resources are distributed among the four construction categories. Resources are defined as metal and energy [see definition of economy in Section 2.5]. The sum of the four values must always equal 100. The *Group Composition* variable determines the number of each unit type required to form a complete attack group, ready to be sent into battle. Finally, the *Initial Economy* is a 2-tuple of integer values. It defines the number of resource producing structures to construct prior to building the factory. The resource producing structures establish an initial economy for

the agent prior to entering combat unit production. The first integer value is the number of metal extractors and the second is the number of solar panels.

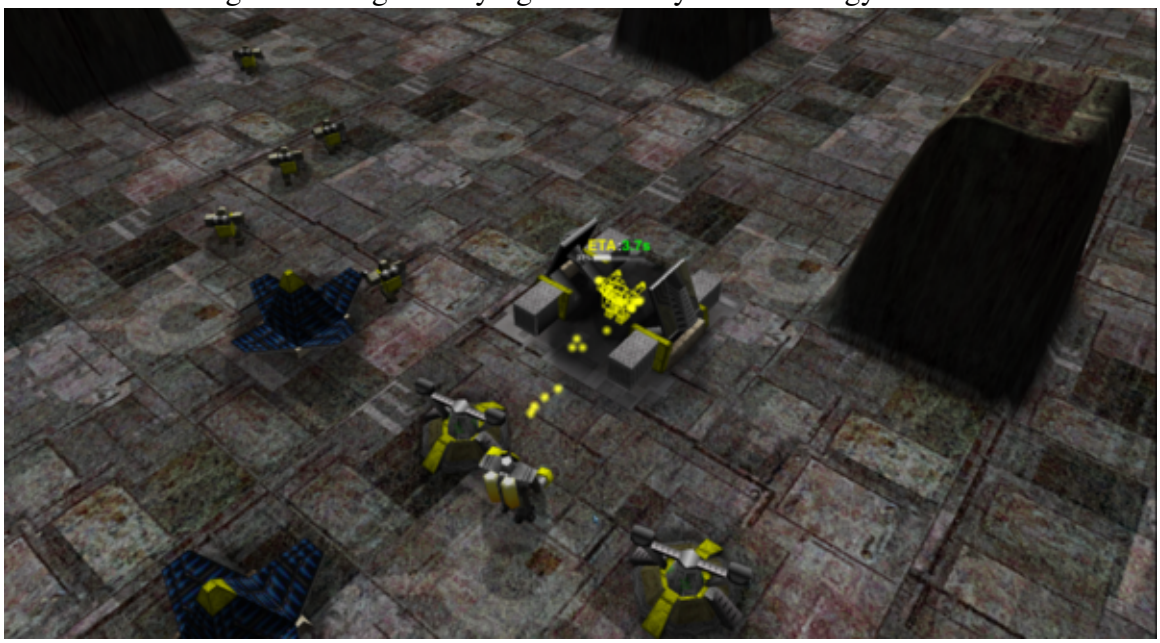
Table 4.1: Strategy Definition Schema

Variable Name	Domain	Description
Build Power	Integer value from 0 to 100	Specifies the percentage of resources used to increase build power.
Economy	Integer value from 0 to 100	Specifies the percentage of resources used to expand the economy.
Defense	Integer value from 0 to 100	Specifies the percentage of resources used to construct defensive structures.
Units	Integer value from 0 to 100	Specifies the percentage of resources used to construct combat units.
Group Composition	Provides a non-negative integer value for each unit type	Gives exact size and composition of an attack group.
Initial Economy	2-tuple of non-negative integer values	Number of metal extractors and solar panels to construct prior to building the factory

4.2.2 Strategies. We select eight general strategies to be used in our agent framework. Previous approaches of RTS AI research have employed two to four scripted opponents for testing [see Chapter 3]. The low number of scripted opponents is due to the difficult and time consuming nature of programming each individual agent by hand. By testing the system with eight strategies, the agent framework effectively behaves as eight different scripted opponents. This is double the number of opponents in previous research. By using this relatively large number of strategies, we validate the usefulness and effectiveness of the strategy schema approach. The following subsections provide English descriptions of each strategy.

4.2.3 Infantry Rush. This is a rush strategy that uses tech level 1 k-bot infantry units as illustrated in Figure 4.1. The intent is to attack the opponent as soon as possible. The attack group consists of three infantry units—one flea, one peewee, and one rocko. A small initial economy is needed to support the production of the infantry units. Only one metal extractor, two solar panels, and a k-bot lab are required for the initial economy. Once they are constructed, all resources are devoted to building infantry units. The agent continues to build waves of three infantry units until the game is over.

Figure 4.1: Agent Playing the Infantry Rush Strategy in BA



4.2.4 Tank Rush. This is a rush strategy that employs level 1 assault tanks. It is depicted in Figure 4.2. The intent is to build an attack group of three assault tanks (stumpy units) and attack as quickly as possible. The initial economy is composed of three metal extractors, three solar panels, and a vehicle plant. Once the initial economy is established, all resources are devoted to building tanks. The agent continues to build waves of three tanks units until the game is over.

Figure 4.2: Agent Playing the Tank Rush Strategy in BA

4.2.5 Blitz. With the blitz strategy, shown in Figure 4.3, the goal is to build a massive army large enough to power through any defenses and destroy the enemy base as quickly as possible. Therefore, the agent creates an initial economy and a vehicle plant as with the Tank Rush strategy, and then begins to build its army of four raider tanks (flash units), one missile truck (samson unit), size assault tanks (stumpy units), and two rocket tanks (janus units). Since this strategy requires a larger army than the rush strategies, it allocates 2% of its resources to build power which helps it construct the army faster. Another 10% of its resources are dedicated to expanding its economy—providing the metal and energy needed to construct the tanks. Since this is a highly offensive strategy, like rush, it does not allocate any resources to defense.

4.2.6 Defended Artillery. This strategy, seen in Figure 4.4, leverages the long range of artillery. It focuses on building attack groups of two artillery trucks (shellshocker units) along with a supporting squad of one scout jeep (jeffy unit), two raider tanks (flash

Figure 4.3: Agent Playing the Blitz Strategy in BA

untis) and two assault tanks (stumpy units). When the attack group is ready, the artillery is moved within range of the enemy base but outside the range of any enemy defenses and the artillery fires on the base. The support squad defends the artillery trucks from enemy units. The agent continues to produce artillery attack groups while expanding its economy until the game is over. Because the artillery attack group is considerably more expensive than the rushing attack groups, more resources are spent on build power and economy to ensure the agent's economy can support the strategy. Since the artillery units provide little defense against enemy units attacking the base, 10 % of the agent's resources are devoted to defense.

4.2.7 Bomber. Using air power, the agent constructs an air force of strategic bombers to destroy the enemy from above. Figure 4.5 offers a screenshot of the strategy in action. An attack group consists of four bomber aircraft (thunder units). The bombers are sent to directly attack the enemy commander doing repeated bombing runs. Using their

Figure 4.4: Agent Playing the Defended Artillery Strategy in BA

speed, the bombers are able to quickly fly into enemy territory, drop their payload and fly out, leaving the opponent only a small window of time which the bombers are in range of the opponent's weapons. In addition to their speed, bombers, like other aircraft, also benefit that most ground units inflict much less damage when attacking air units. Unless the opponent constructs anti-air towers and/or anti-air units like the jethro, the samson, or the freedom fighter, the opponent will be nearly defenseless against the bombers. The agent continues to produce attack groups until the game is over.

4.2.8 Anti-air. The anti-air strategy is especially tailored to counter play styles that rely on air power. It constructs a strong initial economy. To defend against enemy aircraft, 20% of its resources are devoted to defense. The agent constructs four freedom fighters to shoot down any enemy aircraft and five bombers (thunder units) to bomb the commander with.

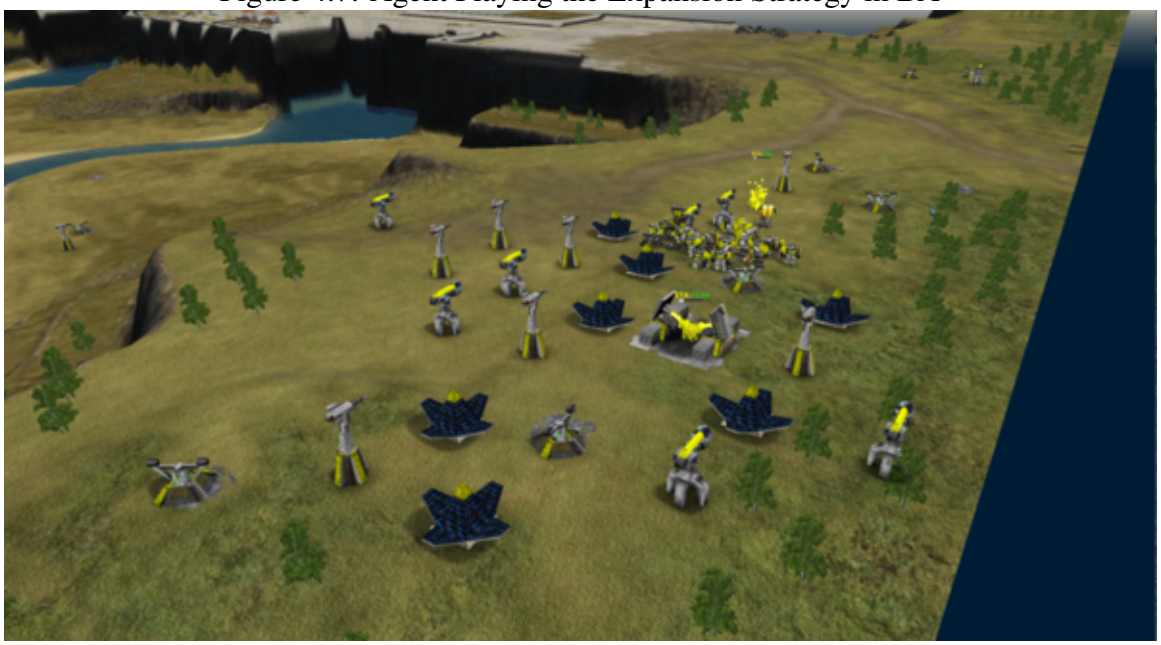
Figure 4.5: Agent Playing the Bomber Strategy in BA

Figure 4.6: Agent Playing the Anti-air Strategy in BA

4.2.9 Expansion. This strategy can be seen in Figure 4.7. It focuses on expanding as quickly as possible to control as much of the map and the available resources as it can.

Therefore, much of its resources are spent on build power and economy so it has the workforce, metal, and energy needed to expand quickly. With only 55 % of its resource dedicated to combat units, the agent slowly builds up an army of one scout (flea unit), four light infantry (peewee units), two rocket infantry (rocko units), four anti-air infantry (jethro units), two plasma canon infantry (hammer units) and eight heavy infantry units (warrior units) to attack the enemy.

Figure 4.7: Agent Playing the Expansion Strategy in BA



4.2.10 Turtle. The turtle strategy, depicted in Figure 4.8, emphasises defense above all. It invests in economy and build power while protecting its investment with heavy defenses. It builds light laser towers and anti-air towers while leveraging its many combat units as mobile defenses. Since it is building a large army of four raider tanks (flash units), three missile trucks (samson unit), eight assault tanks (stumpy units) and three rocket tanks (janus units), the constructed vehicles are available as defensive units for a prolonged period of time while the fist attack group is built.

Figure 4.8: Agent Playing the Turtle Strategy in BA

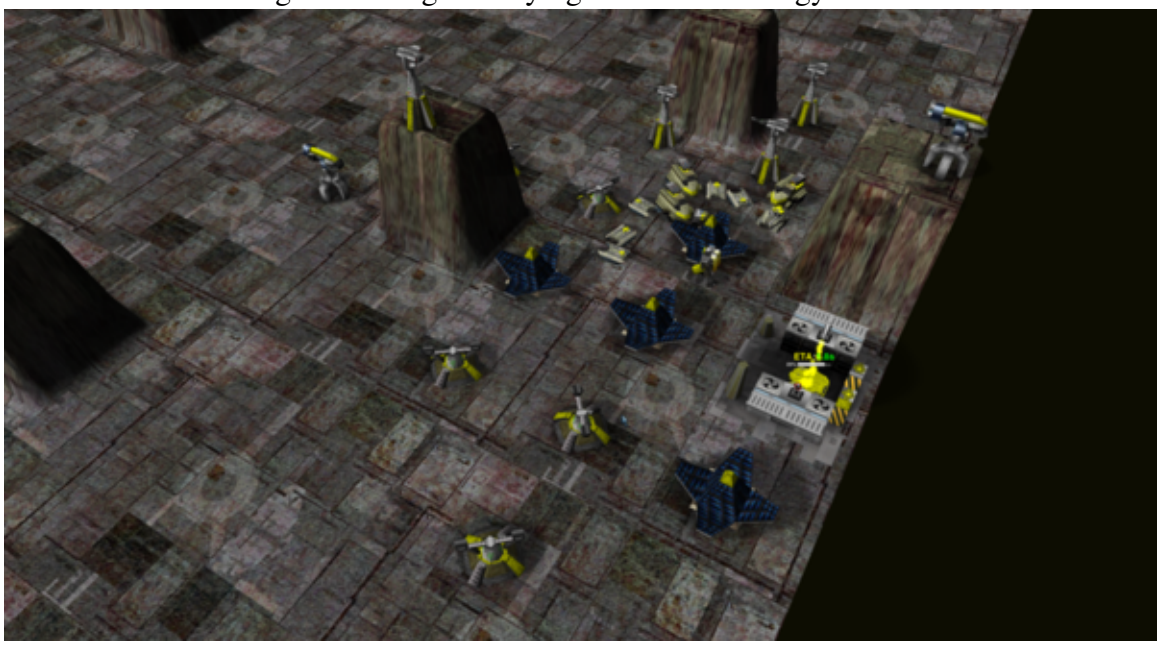


Table 4.2 provides the abbreviations for each of the strategy definition names used throughout the remainder of this document.

Table 4.2: Strategy Abbreviations

Full Name	Abbreviation	Full Name	Abbreviation
Infantry Rush	IR	Tank Rush	TR
Blitz	Blitz	Defended Artillery	Artil
Bomber	Bomber	Anti-air	Antiair
Expansion	Exp	Turtle	Turtle

4.2.11 Strategy Definitions. Table 4.3 defines the strategies in terms of numeric values according to the strategy definition schema presented in Table 4.1.

The numbers under group composition are the count of each unit type. The types, in the order listed above, are: flea, peewee, rocko, jethro, hammer, warrior, jeffy, flash, shellshocker, samson, pincer, stumpy, janus, freedom fighter, banshee, and thunder. Under

Table 4.3: Strategy Definitions

	Construction Variables					
Name	Build Power	Economy	Defense	Units	Group Composition	Initial Economy
IR	0	0	0	100	1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	1 2
TR	0	0	0	100	0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0	3 3
Blitz	2	10	0	88	0 0 0 0 0 0 0 4 1 0 0 6 2 0 0 0	3 5
Artil	5	15	10	70	0 0 0 0 0 0 1 2 0 2 0 2 0 0 0 0	3 4
Bomber	5	20	0	75	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4	3 5
Antiair	5	20	20	55	0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 5	3 5
Exp	10	20	15	55	1 4 2 4 2 8 0 0 0 0 0 0 0 0 0 0	2 3
Turtle	5	10	30	55	0 0 0 0 0 0 0 4 3 0 0 8 3 0 0 0	3 4

the initial economy column, the integers represent the number of metal extractors and number of solar panels to be constructed prior to building the factory.

4.3 Making Strategic Decisions

The agent begins by building the number of metal extractors and solar panels defined in its initial economy. The agent then consults its *Group Composition* variables to determine what factory type it needs and constructs it. Once the factory is complete, the agent exits initial economy mode and enters normal mode.

In normal mode, whenever a construction unit or factory is idle, the agent must decide how to use it. This decision is guided by the four construction variables, *Build Power*, *Economy*, *Defense*, and *Units*. The resources (metal and energy) are spent into the four categories such that the ratio of the resources spent per category divided by the total resources spent matches the construction variable values. When spending on the build power category, more construction units are built. When spending on the economy category, more solar panels and metal extractors are constructed. When spending on the defense category, defensive structures, such as light laser towers and anti-aircraft towers are built. When spending on the units category, combat units are created according to the *Group Composition* variables.

4.4 Groups Tactics

Group tactics can be complex with many difficult sub-problems to solve. Much research has already been accomplished in this area as presented in Chapter 3. Because the focus of our research is strategic-level decision making, we keep the tactics as simple as possible. Future work will integrate the contributions of recent tactical-level AI into our agent framework.

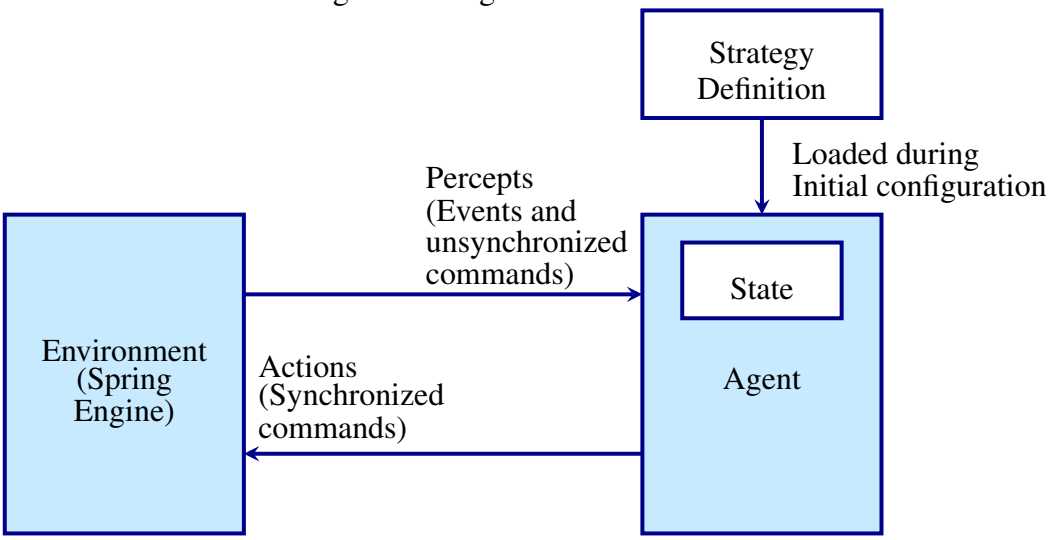
There are two types of groups: attack and defense. All groups begin life as a defense group and eventually become an attack group. An agent only has one defense group, but can have many attack groups. Groups are built in cycles. When a unit is built, it joins the agent's single defense group. The defense group guards the commander and counter-attacks any assault on the base if the attacker is within global line-of-sight (LOS). This keeps the units usefully defending the commander and base while they wait for remaining members of their group to be created. When the defense group is complete—when it has all the units defined by the *Group Composition* variable—it becomes an attack group. A new, empty defense group is created to replace the previous one. The new defense group will receive the next wave of constructed units. Once in attack mode, the group selects a target according to the programmed tactics.

To select a target, the distance from each enemy unit to the group's center of mass is calculated and the closest enemy ground unit is attacked. This tactic was inspired by [50]. Once the current target is destroyed, a new target is selected. This cycle is repeated until all enemy units are destroyed or the attack group is destroyed. Groups containing bomber aircraft have different tactics. Instead, the group always targets the commander regardless of position. Additionally, only ground units defend the base. Aircraft, such as the bomber, do not defend the base because if the do, they can accidentally bomb friendly units and structures as well as the enemy.

4.5 Agent Architecture

Figure 4.9, inspired by Russell and Norvig's discussion of agent structure in Chapter 2 of [34], gives a high-level representation of our agent and its environment. The agent structure most closely resembles a model-based reflex agent with factored state as discussed in [34], pages 50-58.

Figure 4.9: Agent and Environment



4.5.1 Percepts. The agent has two ways it perceives its environment. 1) The game events sent from the Spring engine provide the agent with notification of important events. 2) The agent uses unsynchronized commands to query the Spring engine while processing events. The information the agent can obtain form querying the Spring engine is as follows:

1. Location and type of each friendly unit and building

2. Location and type of each enemy unit and building

3. The agent's amount, storage capacity, production rate and usage rate for both metal and energy.

4. Available construction sites

Table 4.4: Engine Event Table

Event Name	Description
update	This event is fired every frame of the game. The engine executes (approximately) 30 frames per second.
unitCreated	This event occurs when a unit comes into existence, even though the unit may not be functional yet. For example, when the commander begins building a metal extractor, a unitCreated event is fired, even though the metal extractor is still under construction. This event precedes the unitFinished event.
unitFinished	This event indicates construction on a unit is complete and the unit is ready for orders. This event succeeds the unitCreated event.
unitIdle	This event is sent when a unit has no order to execute.
unitDamaged	Whenever a friendly unit receives damage, this event is fired.
unitDestroyed	This event occurs when a friendly unit is destroyed.
enterLOS	Whenever an enemy unit enters the LOS of a friendly unit, this event is fired.
leaveLOS	This event indicates an enemy unit has left the LOS of a friendly unit.
enemyDestroyed	Anytime an enemy unit is destroyed, this event fires.
enemyFinished	This event occurs when an enemy unit has been completely constructed and is ready for its first order.

The game events, along with descriptions of each, are listed in Table 4.4. The *update* event is periodic while all other events are aperiodic.

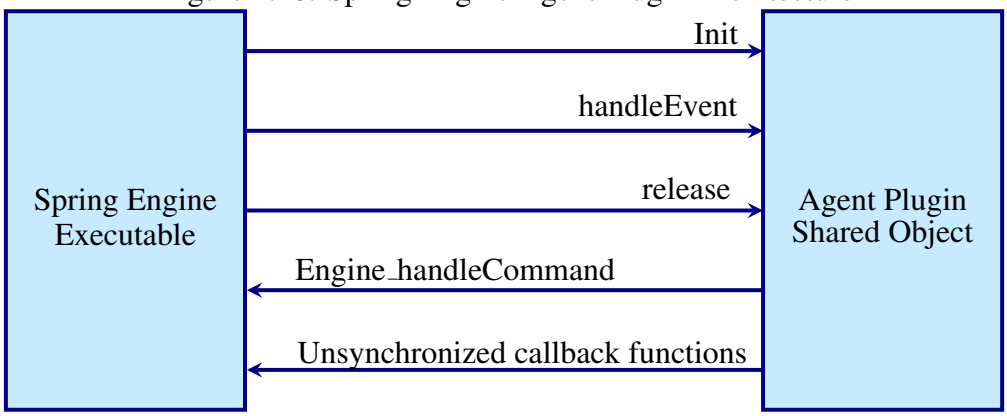
4.5.2 Agent Actions. The agent directs five basic entities, the commander, construction units, the factory, the defense group, and the attack groups. When an event is triggered, the agent assesses the situation and performs actions on the effected units if necessary. For example, if the *unitIdle* event is triggered for the commander, the agent can

either order the commander to assist the factory or construct a new building. The actions the agent can perform are:

1. Commander
   (a) Assist the factory in constructing new units
   (b) Erect a building (econ or defense)
2. Basic construction units always assist the commander in whatever he is building
3. The factory creates units according to the *Group Composition* variable.
4. The defense group is always ordered to guard the commander unless the base is under attack
5. Attack groups must be given a target to attack

4.5.3 Spring Engine AI Interface. The Spring engine uses a plugin interface to dynamically load agents into the game. If a game is directed to use an agent as one of the players, the engine searches for a AIInfo.lua file in the subdirectories of the /AI/Skirmish/ directory that defines a matching agent name. Once it finds a match, it loads the shared library with a name of libSkirmishAI.so from that directory. [43]

Figure 4.10: Spring Engine Agent Plugin Architecture



The Spring engine skirmish AI (agent) plugin interface is defined by four files found in the rts/ExternalAI/Interface/ directory in the Spring engine source code. The four files, shown in Table 4.5, are named AISEvents.h, AISCommands.h, SSkirmishAICallback.h and SSkirmishAILibrary.h. The interface uses the concept of events and commands. Events are signals that the engine uses to call into the agent. They are implemented as data structures that are sent to the agent by calling the agent's handleEvent function. Commands are signals that the agent uses to call back into the engine. Commands come in two forms, synchronized and unsynchronized. Synchronized commands change the state of the game; they include commands that move units, build structures and direct units to attack. Commands are sent over the network to all players to be executed on all simulations. Synchronized commands are implemented as data structures that the agent sends to the engine by calling the engine's handleCommand function. Unsynchronized commands do not alter the game state. They include commands that ask for a unit's position on the map or the player's currently available resources. Unsynchronized commands do not need to be sent over the network. There are 614 different unsynchronized commands, each with its own callback function.

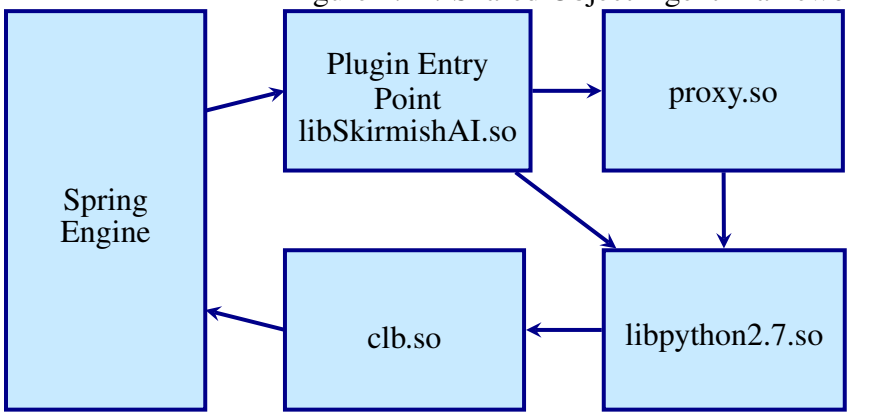
Table 4.5: Spring Skirmish AI Interface Files

Interface File	Functionality
AISEvents.h	Defines event topics and data structures
AISCommands.h	Defines command topics and data structures
SSkirmishAICallback.h	handleCommand, unsynchronized callback functions
SSkirmishAILibrary.h	init, release, hadleEvent, levelOfSupport

Given this interface, every agent must implement its own init, release, and handleEvent functions and make these functions global symbols in the shared library. In order to allow the agent to call back into the engine, the engine passes the memory address

of the callback structure as a parameter to the init function when the engine initializes the agent. The callback structure contains a pointer to the handleCommand function and all 614 unsynchronized callback functions. Thus the agent receives the callback structure and can use it to invoke any of the 615 callback function on the engine.

Figure 4.11: Shared Object Agent Framework



The Spring engine has AI interfaces for C, C++, and Java. Both the C++ and Java interfaces are implemented on top of the basic C interface. We chose to use the C interface as it is the most simple and straight forward of the three. During initial framework design, the Java interface was tested but found to have some software bugs. Most of our agent is written in Python, with only a minimal amount of glue code in C to communicate with the Spring engine. Figure 4.11 presents the relationship between the different shared objects in our agent framework. A shared object is the Unix equivalent of a dynamically-linked library on Windows. The arrows represent the direction of the function invocations. The *proxy.so* library configures the environment and loads *libpython2.7.so* which contains the Python interpreter. Once the Python interpreter is loaded into memory, the *proxy.so* loads the *agent.py* module and returns a reference of the *Agent* class to *libSkirmishAI.so*. The *libSkirmishAI.so* then creates instances of the *Agent* class for each player. Whenever events are sent from the Spring engine, the *libSkirmishAI.so* calls the appropriate function

on the receiving agent. If the agent needs to call back into the Spring engine, it uses the *clb.so* c library to invoke the required function on the Spring engine. The agent interfaces with *clb.so* via the ctypes library [32].

This design enables us to use the high-level language, python. The Python developers created the language to be very readable and expressive. The design of Python includes functions as first-class objects, dynamic typing, partial functional programming support, and built-in syntax for dictionaries and lists. These features give us the ability to write the framework in fewer lines of code than would be required if using a lower level language like C++ or Java. This saved a great deal of development time and allowed us to write the agent framework from scratch in under four months!

4.5.4 Agent Design. The agent framework is composed of 16 Python classes. They are presented in Figure 4.12. The diagram provides a general understanding of each objects role, including its attributes and methods. Figure 4.13 illustrates the associations between these classes using a UML class diagram. The addition of the associations presents a general understanding of the static relationship between the classes.

Figure 4.12: Classes of Agent Framework

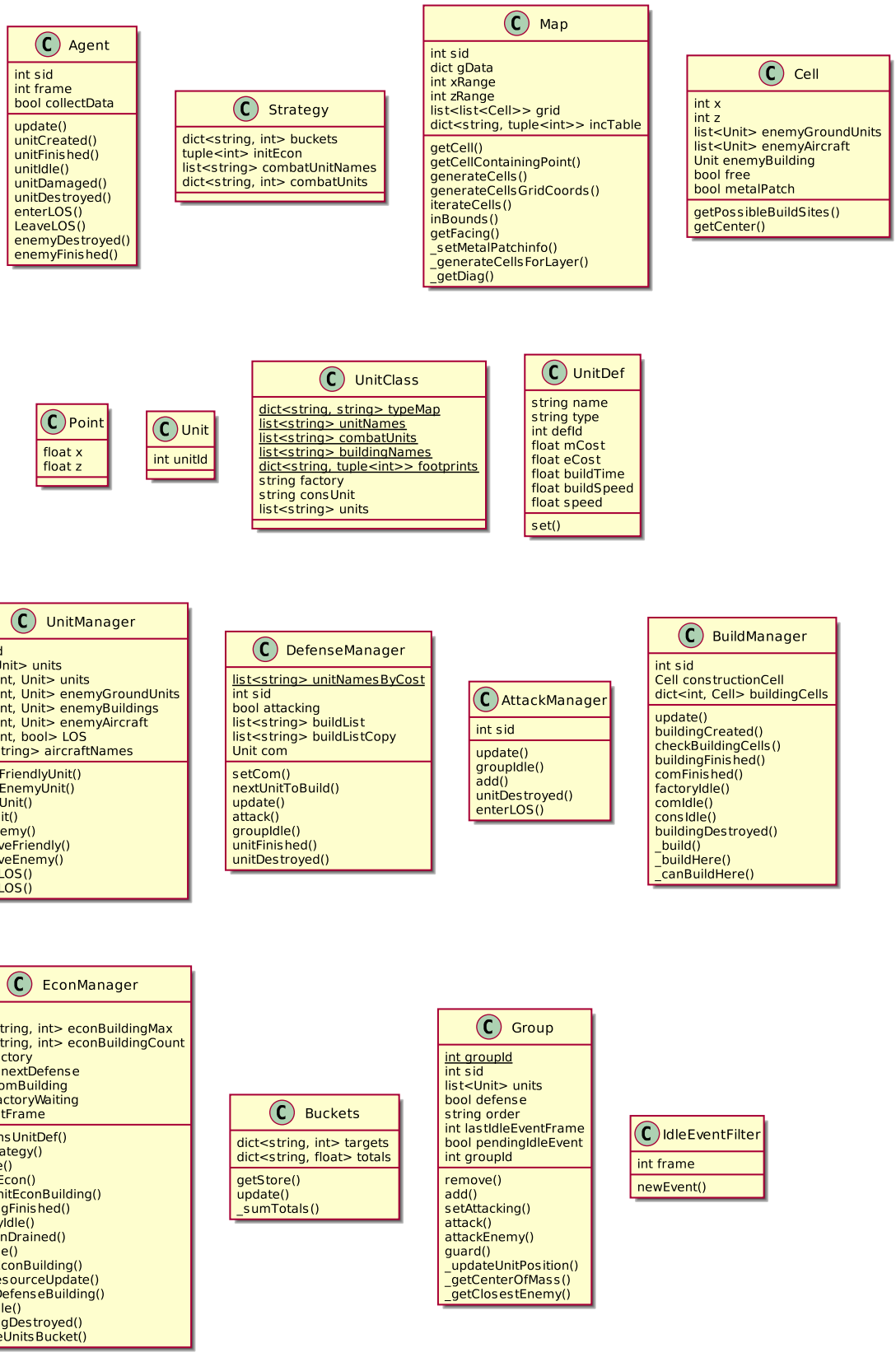
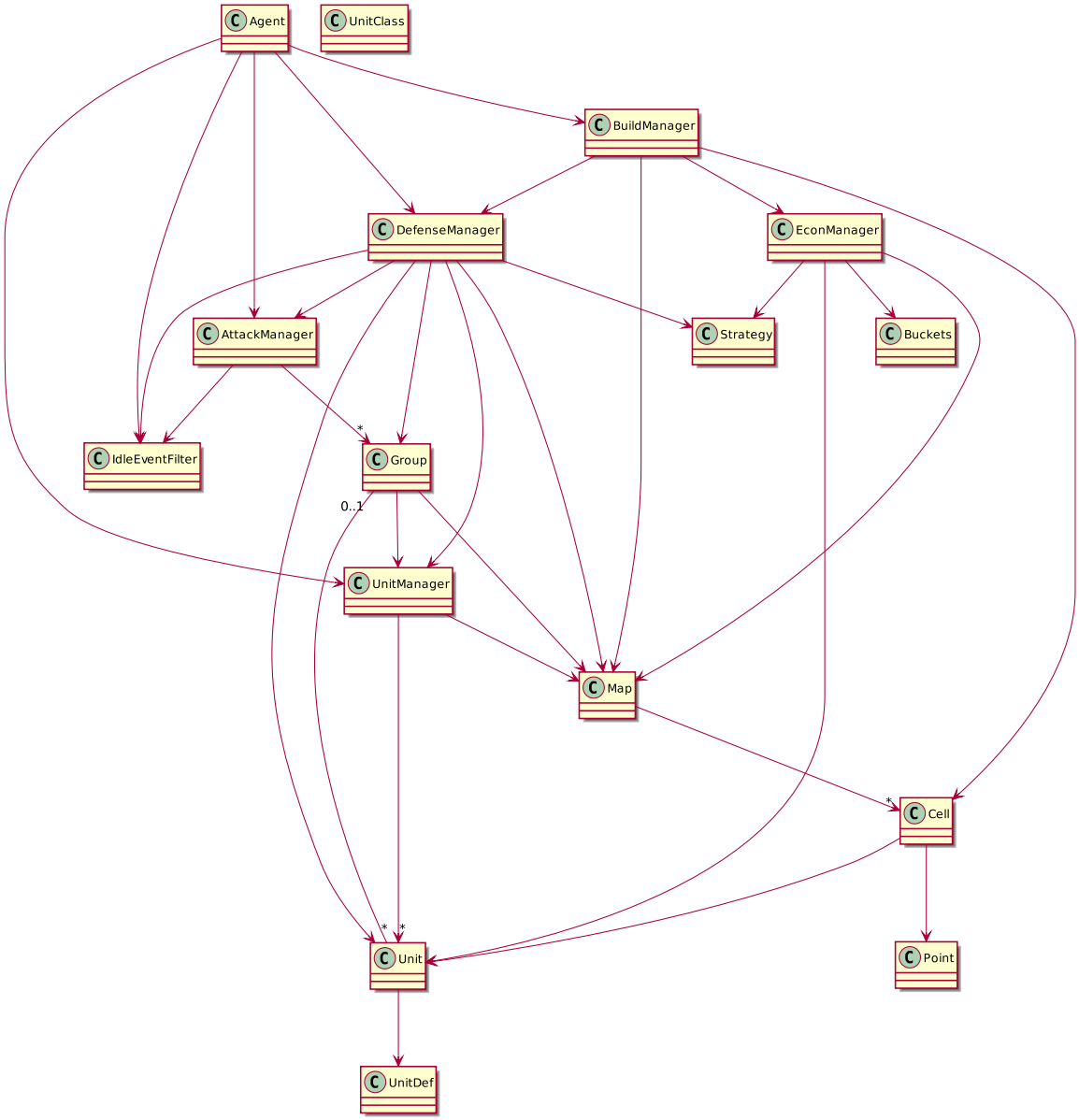


Figure 4.13: Class Diagram



Python classes can be organized into modules which can also contain attributes and functions. The modular structure for organizing classes and their relationships cannot be modeled with simple UML class diagrams. To capture the relationship between modules and classes, Figure 4.14 was created. The eight modules correspond to the eight *.py* files

in the agent framework source code. The arrows in the figure denote dependencies between modules—in other words, the module the arrow originates with imports the module the arrow terminates at. Classes defined in a particular module are displayed inside that module. Also, UML notes are used to list any attributes and functions defined in the module.

Figure 4.15 shows the use case diagram of our agent. The Spring Engine is an actor outside our agent system. The use cases map one-to-one with the events listed in Table 4.4. The update use case is periodic while all other use cases are aperiodic. Sequence diagrams of each use case are provided in Appendix A for a more detailed understanding of the interactions between the software objects.

Figure 4.14: Package Diagram

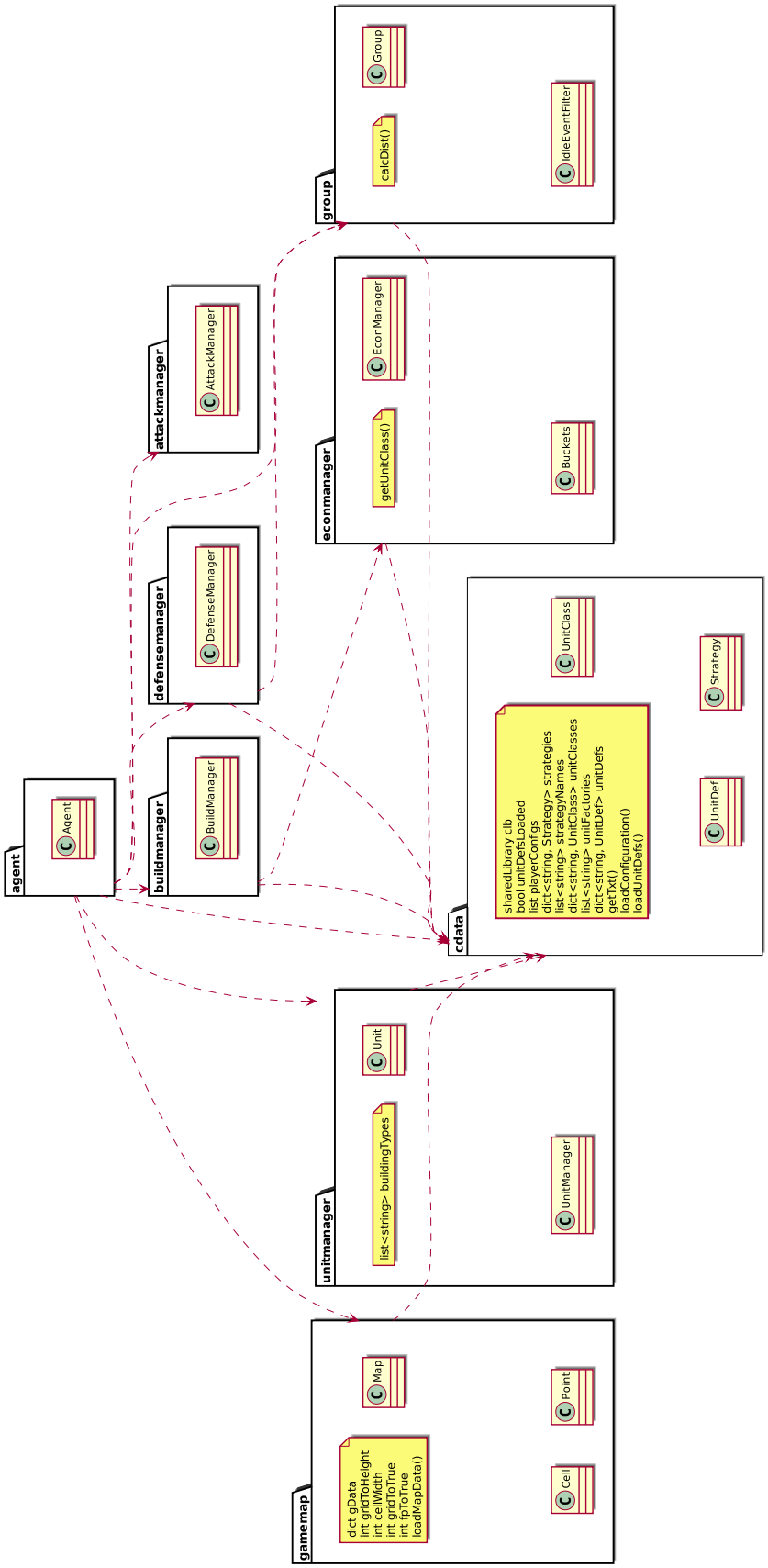
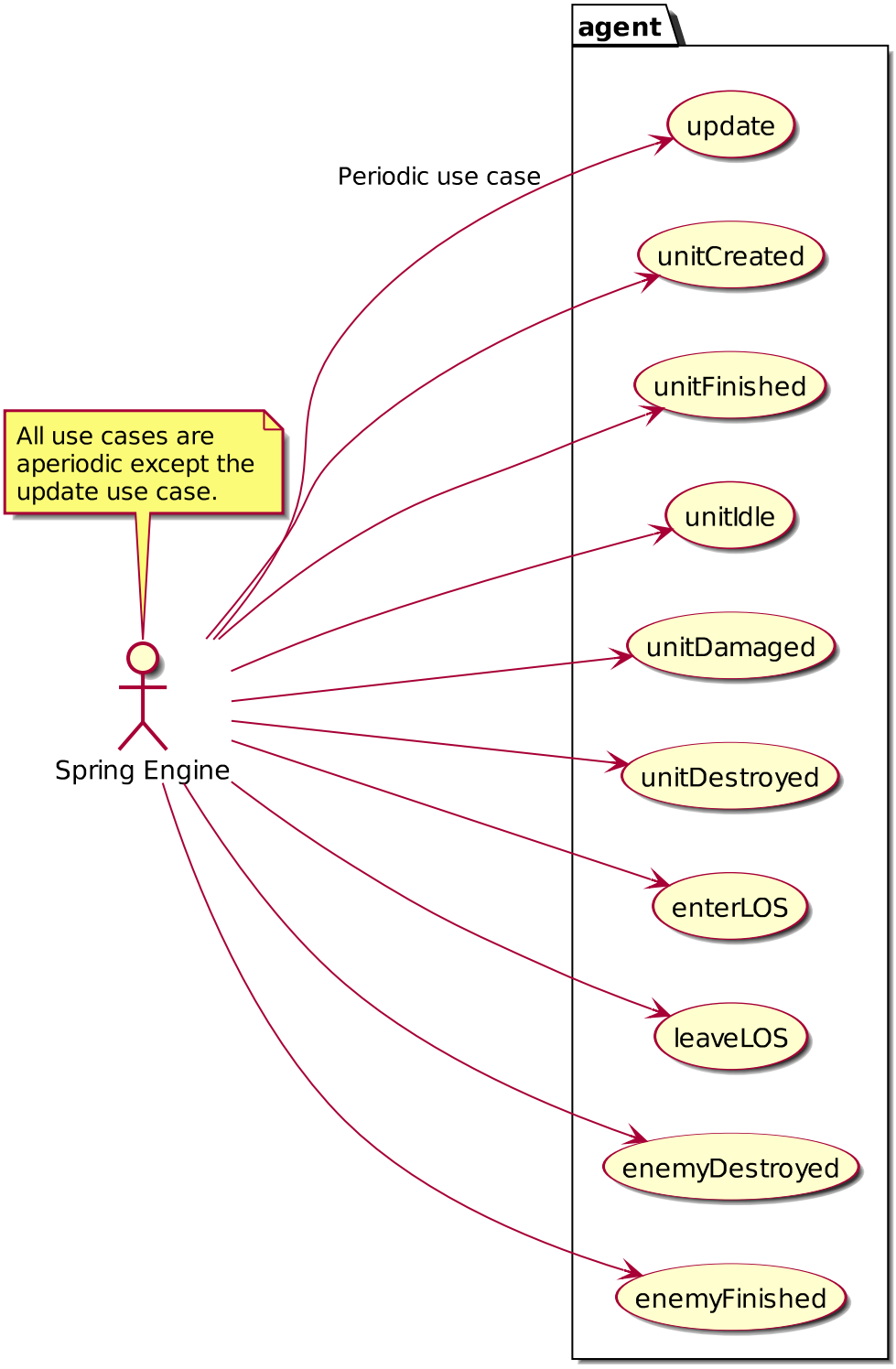


Figure 4.15: Use Case Diagram



4.5.5 Grid world. The map dimensions in game units are 4,096 x 4,096 for the small map, 6,144 x 7,168 for the medium map, and 10,240 x 10,240 for the large map. To reduce the decision space for the agent, a grid world concept was applied. We partition the map into square cells. A cell in the grid is 128 game units x 128 game units. Cells of this size are large enough that any building may be constructed on it while still allowing room for units to navigate. This data structure, along with its supporting methods, are defined in the Gird class in the gamemap.py file. The grid is used for building placement and target selection. Using a single data structure for both algorithms reduces code duplication and improves readability.

We implement building placement and target selection by searching for the closest cell that meets certain criteria. For building placement, the cell must be free of any other construction, and must be suitable for the building type. For target selection, the cell must contain an enemy unit. Both these problems are solved with the help of a Python generator [31]. The generator's input parameters are the coordinates of a starting cell. It yields the cells in order of closest to furthest from the starting cell. Generators provide a memory-efficient method for producing large lists as the items are computed lazily—only when they are needed. We chose to use Manhattan distance as we prefer the trade-off in increased execution speed and a simplified algorithm over the loss of accuracy when compared to using euclidean distance. This trade-off is acceptable as our research objectives requires a real-time agent.

The generator is used by both the building placement and target selection algorithms. The algorithms only need to implement their respective test criteria to test the generated cells. They are not concerned with the details of distance from the starting cell, as this knowledge in encapsulated in the Grid class.

4.6 Constraints

Perfect Information. The agent framework operates in a perfect information environment. A cheating AI is appropriate for military training and education, as it does not hinder the training. The agents can still display strategic decision making while using perfect information. On the other hand, using imperfect information, also known as fog of war in RTS [42], would require implementing a scouting module in the agent framework, which would be very time consuming. We choose instead to focus our effort in implementing strategic behavior. Future work will integrate a scouting module into the framework to allow the agent to operate in an imperfect information environment.

Unit Types. Balanced Annihilation (BA) contains over 317 unit and building types. In order to keep our framework manageable, we limit the number of unit types to 27. The unit types the framework supports are as follows:

1. *Combat units*: The agent supports 16 combat units. They are flea, peewee, rocko, jethro, hammer, warrior, jeffy, flash, shellshocker, samson, pincer, stumpy, janus, freedom fighter, banshee, and thunder. The pincer and banshee unit types are not employed in any of the eight strategies.

2. *Construction units*: The agent supports four construction units. They are commander, construction k-bot, construction vehicle, and construction aircraft.

3. *Building units*: The agent supports seven building types. They are k-bot factory, vehicle factory, aircraft factory, metal extractor, solar panel, light laser tower, and anti-air tower.

Including support for additional units is straightforward. The text files config/buildings.txt and config/unitClasses.txt, located in the agent framework source code directory, are human-readable ASCII text files and can be edited to include additional units.

Naval Warfare. We do not include naval warfare as the Air Force's interests are more toward air and land. For this reason, the agent cannot play games on water maps.

4.7 Advancement of AFIT Research in RTS AI

This work is a continuation of the research effort [50] from the Air Force Institute of Technology (AFIT). We have expanded the work from a land-only domain to include the air-power domain as well. The number of combat units the agent supports increased from five unit types to sixteen unit types. The addition of air-power and increase in unit types allows for a larger variety of strategies. The unit tech level is restricted to tech level 1 in order to create more balance among the units so no one unit dominates the game. The agent framework can now play on many different maps; the previous agents could only play on metal maps and were only tested on a single metal map. The maps employed for testing have actual metal extraction spots. The use of these maps is more consistent with actual BA game-play. We doubled the number of static opponents from four to eight. Having several strategies is important as our focus is on the strategic level of gameplay. Furthermore, additional opponents may be created by simply defining a new strategy definition. Previously, to create a new opponent, one had to write an entire script by hand. Our strategy definition approach is scalable and extensible. The first approach required training one classifier per opponent. We are able to train a single classifier for all eight opponents. Because our agent framework may be used as a base to implement future dynamic agents, both the static agents and the dynamic agents will share the same tactical algorithms. This allows an objective evaluation of the dynamic agent's strategy decision-making performance, when tested against the static agents, as they all share a common tactical base.

4.8 Balanced Annihilation

The BA game was chosen as the platform used to implement our framework. Our research is focused on strategic AI and not on low-level AI. Acceptable unit-level AI is built-into BA. Thus we can put our effort into implementing the higher-level AI and leave the unit-level AI to the BA default implementation. BA is a popular game in RTS AI research, therefore, there already exist a body of work to build on. Starcraft is also very popular as an RTS AI research platform. However, BA was chosen for several reasons. Starcraft is 2D only, while BA is 3D. Since our vision is for CGF in military training, the additional realism of a 3D environment is preferred. Additionally, Starcraft's AI API enforces fog of war—imperfect information. We desired a perfect information environment to avoid implementing a scouting module, thus, again BA is the better choice. Furthermore, BA is open source and multi-platform while Starcraft is closed source and Windows only. So BA provides more flexibility in modifying the engine and porting the game to new systems. BA supports a larger scale in terms of both number of units and map size.

For these reasons we claim BA is a good platform to use in attaining our research goal.

4.9 Chapter Summary

This chapter presents our strategy definition schema, formulates eight strategy definitions, and explains the design and implementation of our agent framework. With the agent framework methodology and design defined, the second half of our methodology, strategy classification, is addressed in the next chapter.

5 Real-time Strategy Classification

5.1 Introduction

In the previous chapter, the methodology is explained for achieving the fist three objectives: defining a strategy schema, formulating strategy definitions, and creating an agent architecture that can run strategy definitions. We now turn our attention to the methodology used to achieve the last three objectives: generating a high-quality data set, creating real-time strategy classifiers, and discovering appropriate counter strategies for each map. To accomplish these objectives, we run the strategies against each other on all three maps over several iterations and periodically collect game state observations (unit types, unit positions, metal and energy statistics). The data set, along with AI techniques, is used to create classifiers which can classify a given opponent game state observation into one of the pre-defined basic strategies. This is done by learning the strategy centers or boundaries in game state observation space. We then find appropriate counter-strategies for each basic strategy based on the wins and losses of the games played. There are four steps to creating our classifiers.

1. Data collection
2. Pre-process the raw data set into a more useful form. This involves four sub-steps:
   (a) Transform the features into a more practical form.
   (b) Drop the first 32 samples of each game.
   (c) Using the new data set, create two data sets, one with economy features (Econ) and one without (No Econ).
   (d) Normalize the two data sets across features.
3. Perform feature reduction. Three different approaches are employed:

(a) Do nothing and use the data set as is.

(b) PCA

(c) LDA

4. Train the classifier. We use two different algorithms:

(a) K-means clustering plus K-NN algorithm

(b) SVM algorithm

These steps are explained in more detail in the following sections. The final section of this chapter discusses the discovery of counter strategies from the data set.

To implement the tasks Python [46] scripts were written that leverage the SciPy [15] and NumPy [26] libraries. NumPy is a numerical package which provides efficient implementation and manipulation of matrix and array data structures via the internal use of LAPACK. SciPy builds on NumPy's features in order to provide a MATLAB-like environment. Implementations of PCA, LDA, k-means, K-NN and SVM are provided by the machine learning SciPy extension library *scikit-learn* [36]. The learning algorithms provided by scikit-learn are written as classes which expose a consistent object API. This allows us to abstract our Python test scripts to easily use different types of feature reduction and classification techniques.

5.2 Data Collection

We play the eight strategies against each other for 20 iterations. Strategies are not matched-up against themselves. Thus, there are $\frac{7\times8}{2} = 28$ different match-ups. Data collection is performed on three different maps: a small map, a medium map, and a large map. Using three differently sized maps exposes the strengths and weaknesses of the different strategies. Additionally, using fewer maps would likely produce biased experiment results. The maps were selected from popular maps freely available for

download from *springfiles.com*. The actual map names are provided in Table 5.1. Each match-up is repeated 20 times, therefore a total of (3 maps)·(28 match-ups)·(20 repetitions) = 1,680 games were executed. During each game, we collect a game state snapshot every 5 seconds (once every 150 game frames or cycles). Each agent is responsible for collecting its own data. A game state snapshot for an agent consists of the following items.

1. Frame number (a measure of time) of the match when the state snapshot is taken
2. The type and (x, z) position of each of the agent's units and buildings
3. Amount, storage capacity, production rate and usage rate of both metal and energy

Additionally, at the end of a game, the agent records whether it won or lost the game.

Table 5.1: Map Names

Small	Medium	Large
ThePass	Eye_Of_Horus_v2	DeltaSiegeDryRevX_v3

The samples collected across all the games constitutes our raw data set. We also save the game replay file for each of the 1,680 games along with the command line output. The game replay files record all the actions taken in the game and allows any individual to replay any of the game match-ups on the spring engine with the BA game. The command line output records any special messages sent by the spring engine. This allows future researchers to view exactly what transpired in any of the games, if needed, to validate any assumptions or answer any questions related to game execution. This gives future researchers more confidence in using our data set instead of generating new data and it makes our data set more accessible and useful to a larger audience.

Table 5.2 presents the total execution time for all the 560 games on each map as well as the average time per game, per map, in both real-world time and virtual game time. The

larger the map, the more time it takes to play a game as the combat units have longer distances to travel and the processor has more to compute (i.e. longer trajectories, more obstacles, etc). The virtual time and real-world time differ because the game simulation is run at maximum speed. Even though a game frame means 33.3 ms of game time has elapsed, if the spring engine has completed computing the simulation for the current frame, instead of sitting idle until a full 33.3 ms elapses, it immediately begins to simulate the next frame. The speedup from running the spring engine at maximum speed is included in the table. This allows us to execute all 1,680 games in under 15 hours instead of the 15 days 21 hr and 36 min it would have taken if running the game simulations at normal speed. With the production of the raw data complete, the next step is to pre-process it.

Table 5.2: Data Collection Times

	Small	Medium	Large
Overall Time	2 hr 59 min 00 s	4 hr 57 min 00 s	6 hr 59 min 00 s
Time per Game (Real-world)	19 s	32 s	45 s
Time per Game (Virtual)	10 min 01 s	14 min 03 s	16 min 49 s
Speedup	31.6	26.3	21.3

5.3 Data Pre-processing

The raw data must be transformed into a more useful form. The data set has an entry for the type and (x, z) position of each unit. This creates a variable number of features from sample to sample. Instead, we desire a constant number of features. Therefore, the number of units for each unit type in a sample are counted. So our new data set consists of samples with the following features.

1. The number of units in play of each unit and building type (24 variables)
2. Amount, storage capacity, production rate and usage rate of both metal and energy (8 variables)

Our agent framework spends the beginning of a game creating its initial economy which involves constructing metal extractors and solar panels. This happens regardless of the strategy definition in play (although the strategy definition does control how many metal extractors and solar panels are built for the initial economy). Therefore, the first game observations for any of the 8 strategies is almost identical. Trying to classify the strategy at the beginning of the game is useless because the agents behave very similarly. Because of this, we drop the first 32 samples from each game. This equates to the first 2 minutes and 40 seconds of the game. Around that time, most strategies have completed their initial economy and have begun constructing combat units. At this point, the agents begin to differ in their behavior based on their respective strategy definition.

Two data sets are created; one with economy features (Econ) and one without (No Econ). The data set with economy features has 32 total features: 24 unit type features and 8 economy features. Thus the data set without the economy futures only has 24. There are three possible outcomes of including economy features.

1. The economy features may increase the separation of the strategy classes, thus improving the classifier accuracy.

2. They may contribute little to the separation of the classes, and in turn, have little impact on classifier accuracy.

3. They may be high-variance and chaotic data, thus decreasing the classifier accuracy.

Creating a high-accuracy classifier is one of the research objectives. Since it is unknown which data set produces the highest accuracy classifier, both data sets are used in training and testing the classifiers.

We normalize values for the two data sets across features to give the features equal weight when training the classifiers. To do this, the data for each feature is linearly mapped to a range of -1 to 1. So, for a specific feature, the minimum value for the feature

maps to -1 and the maximum value for the feature maps to 1 and all other values of the feature are linearly scaled between -1 and 1.

5.4 Feature Reduction

It is often desirable to reduce the number of features per observation in a data set prior to training a classifier. The purpose being to reduce the computational overhead of additional features that do not contribute much information to the classifier-generating algorithm. Another possible benefit of performing feature reduction is to remove features that are so chaotic or noisy that they actually deteriorate the accuracy of the trained classifier. Feature reduction is also known as dimensionality reduction. [14, 17]

The feature reduction methods we utilize both have the same general steps. First, train the feature reduction model using only the training data set. Then transform the training data set using the feature reduction model. This results in a data set with fewer features. Use the transformed training set to train the classifier. Prior to classifying an observation with the trained classifier, the observation must also be transformed with the original feature reduction model so the observation is in the same coordinate system as the trained classifier.

5.4.1 Principal Component Analysis. We apply PCA prior to training the classifier. PCA is an unsupervised feature reduction technique—it does not consider the class labels of the samples in the training data set. The feature reduction model of PCA is simply the eigenvectors of the training data set sorted in descending order by eigenvalue. The training data set is transformed into a new coordinate system by simply performing the dot product of the training data set with the eigenvector matrix. Because the eigenvalues are proportional to the variance contained in the transformed dimensions, the first dimension (feature) contains the most variance of the data set, the second dimension (feature) contains the second most variance, and so forth. We chose to keep the first n

features and throw away the rest—where the first n features contain 95% of the variance (computed using the eigenvalues). More details on PCA are available in [14, 17].

5.4.2 Linear Discriminant Analysis. Alternatively, we also apply LDA to our data sets. LDA is a supervised feature reduction technique—it takes into account the labels of the training samples. Like PCA, LDA attempts to find a linear combinations of the features that separates the two classes. Since the data set has eight classes, LDA is performed seven times for seven one-vs-rest binary analysis resulting in a data set with only seven total features per observation. For more discussion on LDA, see [14, 17].

5.5 Building a Classifier

Using the processed training data set, classifiers are created to classify real-time game observations as strategies. The classifier is important as it is the component that will give future dynamic strategy agents the ability to identify its opponent's strategy. The process of training a classifier is in fact a search for class centers or inter-class boundaries depending on the approach. We employ two classifier training methods, k-means with K-NN, and SVM. These classifier algorithms were selected for their complementary strengths and weaknesses. K-means clustering plus K-NN produces a fast executing classifier. However, it has difficulty dealing with non-separable, overlapping classes especially since k-means is unsupervised and it forces the clusters to have roughly equal variance. SVM, while usually slower executing, is better suited for dealing with non-separable classes because it is a soft-margin classifier. [13, 14, 17]

5.5.1 K-nearest Neighbor on K-means. Our fist method uses use a two-step approach to create classifiers. Step 1 involves clustering the data set per strategy and step 2 uses the cluster centers as training points for the actual classifier.

K-means Clustering. First, we divide the training data set into game states for each specific strategy. This produces eight groups of data; each data group has all the player

state snapshots of its respective strategy. The game states for each group are clustered using k-means clustering with k=7. The new clusters are labeled with the group's strategy. This process is repeated for all eight groups of data. The 56 (8 strategies * 7 clusters) points in observation space become our labeled strategy classifier.

K-nearest Neighbor. Classification is performed by using the 56 points as the training samples for a K-NN classifier. We use one-nearest neighbor with euclidean distance as our metric. Thus, to classify a game state snapshot, the distance from the snapshot to each of the 56 points is calculated, and the distances are sorted. The label of the point closest to the sample snapshot is the classifier's predicted strategy.

More information on the k-means and K-NN algorithms is available in [14, 17].

5.5.2 Support Vector Machine. We create a second classifier using a SVM. SVM is a supervised classification technique. It falls within the family of maximum margin classifiers. That is, it attempts to find hyperplanes that maximize the margin between two classes.

SVM is an inherently binary classifier, however it can be extended to a multi-class case by computing pairwise SVM classifiers of all the classes (requires $\frac{C(C-1)}{2}$ classifiers) or computing one-vs-rest SVM classifiers for all the classes (requires $C - 1$ classifiers) where C is the number of classes (or strategies in our case). In our work we use one-vs-rest multi-class SVM in addition to the kernel trick with a Gaussian radial basis function kernel [13]. A more comprehensive SVM treatment is given in [14, 17].

5.6 Learning Counter-Strategies

We use our game win-loss outcomes to find the best counter-strategies for each strategy. This provides the knowledge to choose a good counter-strategy given the opponent's strategy. The best counter-strategy is the strategy with the highest win ratio with respect to the strategy of interest. Thus, for each strategy, we count the number of

wins and subtract the number of losses when played against the other strategies. The values are sorted in decreasing order and the strategy related to the lowest value is the most effective counter-strategy.

5.7 Chapter Summary

This chapter presents the methodology used to achieve three research objectives: generate a high-quality data set, create real-time strategy classifiers, and discover appropriate counter strategies for each map. To accomplish these objectives, we run the strategies against each other on all three maps for several iterations and periodically collect game state observations. The data set along with AI techniques were used to create strategy classifiers. The data set was also employed in discovering counter-strategies. This concludes our second chapter on methodology. The next chapter covers the design of experiments for the implemented agent framework and classifiers.

6 Design of Experiments

6.1 Introduction

This chapter reveals how our classifiers are tested using 5-fold cross-validation, discusses the objectives of our experiments and explains why they are important. It also describes how the results of the classifier tests are processed to show that the objectives of the experiments have been met. The objectives of the experiments, inspired by [7], are as follows:

1. Show the effectiveness of our agent framework in implementing the eight basic strategies.

2. Compare economy and non-economy data sets in terms of accuracy.

3. Compare original features with the two feature reduction techniques (PCA and LDA) in terms of accuracy.

4. Compare K-NN and SVM classifiers in terms of accuracy.

5. Find the best combination of data set, feature reduction technique and classifier type in terms of accuracy.

6. Find the best combination of data set, feature reduction technique and classifier type in terms of speed of execution.

7. Show both the most accurate classifier and the fastest executing classifier have the following properties:

   (a) The standard deviation of the classifier's accuracy is under 5% of the mean of the classifier's accuracy.

(b) The largest delta in accuracy per iteration is less than 5% for each of the three maps.

(c) The producer accuracy is equal to or greater than 75% for every strategy.

(d) The execution speed is under 10% of a game cycle period (a game cycle period is 33.3 ms).

(e) The accuracy is above 80% for over 90% of the time.

The remaining sections of this chapter cover the wins-losses and counter-strategies processing, the classifier testing process, the results processing to reveal the classifier accuracies, the results processing to reveal the classifier execution speeds, and finally, the results processing to reveal the properties of the most accurate classifier and fastest executing classifier. The Wins-losses and Counter-strategies section relates to Objective 1. The Classifier Accuracy section deals with Objectives 2-5. The Classifier Execution Speed section covers Objective 6. The Objective 7 is explained in the last section, Properties of Most Accurate Classifier and Fastest Executing Classifier.

6.2 Wins-losses and Counter-strategies

We desire to meet Objective 1: show the effectiveness of our agent framework in implementing the eight basic strategies. This objective is important because it validates the strategy schema concept, our formulation of our eight strategies using the strategy schema, and most importantly, it provides some means of demonstrating our agent framework implementation is actually able to play the BA game according to a given strategy definition.

Win-loss tables and a counter-strategy table are generated to demonstrate the fulfillment of Objective 1. Three win-loss tables are created; one for each of the three maps. We played 1,680 games—560 per map. So for the 560 games of given map, the wins and losses are counted for each strategy match-up. The losses are subtracted from

the wins, producing a single number per match-up. Since we repeat each match-up 20 times, the values range from -20 to 20. For an explanation on how the counter-strategy table is generated, see Section 5.6.

Although Objective 1 is somewhat subjective, we provide a list of measurable requirements in Table 6.1 to ascertain whether or not the objectives have been met. These requirements are based on the fundamental RTS concept that no strategy dominates all the time in every situation. If this were the case, RTS games would be very boring, because once a player has found the dominant strategy, all players would continually use it, turning the game into a game of luck where human intelligence has no impact on the outcome of the game. Proper formulation of the strategy schema, good choices in strategy definitions, and intelligent design decisions in the agent framework would produce a balanced set of strategies that all have different strengths and weaknesses and where no one strategy dominates but all strategies are good choices in certain situations.

Table 6.1: Objective 1 Requirements

	Requirement
1	No strategy dominates all other strategies on any map.
2	Every strategy has an overall winning record against at least one strategy on at least one map.
3	Every strategy is the best counter-strategy for at least one strategy on at least one map.
4	IR and TR favor smaller maps.
5	Blitz, and Artil favor medium maps.
6	Bomber, Antiair, and Turtle favor large maps.
7	Antiair dominates Bomber.

Requirements 1-3 ensures the agent framework plays each strategy in such a way that every strategy is useful and valuable. Since IR and TR are rush strategies, they should favor smaller maps—hence the need for Requirement 4. The Blitz and Artil strategies have medium initial economies and medium sized armies. For this reason, we expect these strategies to favor medium maps. Requirement 5 ensures this is true. Requirement 6 deals

with strategies that should favor large maps. Specifically, Bomber, Antiair and Turtle, because they all have large initial economies and produce large and expensive armies.

6.3 Classifier Testing using 5-Fold Cross-validation

We have 5 dimensions of test configurations: maps, data sets, feature reduction techniques, classifiers, and cross-validation iterations. The possible values for each dimension are as follows:

- 3 Maps: Small, Medium, and Large
- 2 data sets: with economy features (Econ) and without economy features (No Econ)
- 3 feature reduction techniques: None, PCA, and LDA
- 2 Classifiers: K-NN on K-means and SVM
- 5-fold cross-validation: the experiment is repeated 5 times, each time holding back a different partition of the data set from the classifier training to be used for testing.

For each combination of the testing dimensions, we perform an experiment. An experiment consists of training and testing a classifier. The classifier is trained using only the prescribed training data set and testing is performed with the remaining testing data set. In total we perform (3 Maps) ·(2 data sets) ·(3 feature reduction techniques) ·(2 Classifiers) ·(5-fold cross-validation) = 180 experiments.

Table 6.2: 5-fold Cross-validation

	Data Set Partitions				
	1	2	3	4	5
Iteration 1	Test	Train	Train	Train	Train
Iteration 2	Train	Test	Train	Train	Train
Iteration 3	Train	Train	Test	Train	Train
Iteration 4	Train	Train	Train	Test	Train
Iteration 5	Train	Train	Train	Train	Test

Table 6.2 illustrates the concept of 5-fold cross-validation used to perform our experiments. We partition our data set into 5 sections. When an experiment is run, the iteration of the 5-fold cross-validation dictates which data partition is held back from the feature reduction and classifier training processes to be used to later test the classifier on. Thus, for the K^{th} iteration, the K^{th} partition becomes the testing set and the remaining partitions combine to become the training set. This is explained further in Section 7.10.1 of [17] page 242.

Figure 6.1: Experiment Process

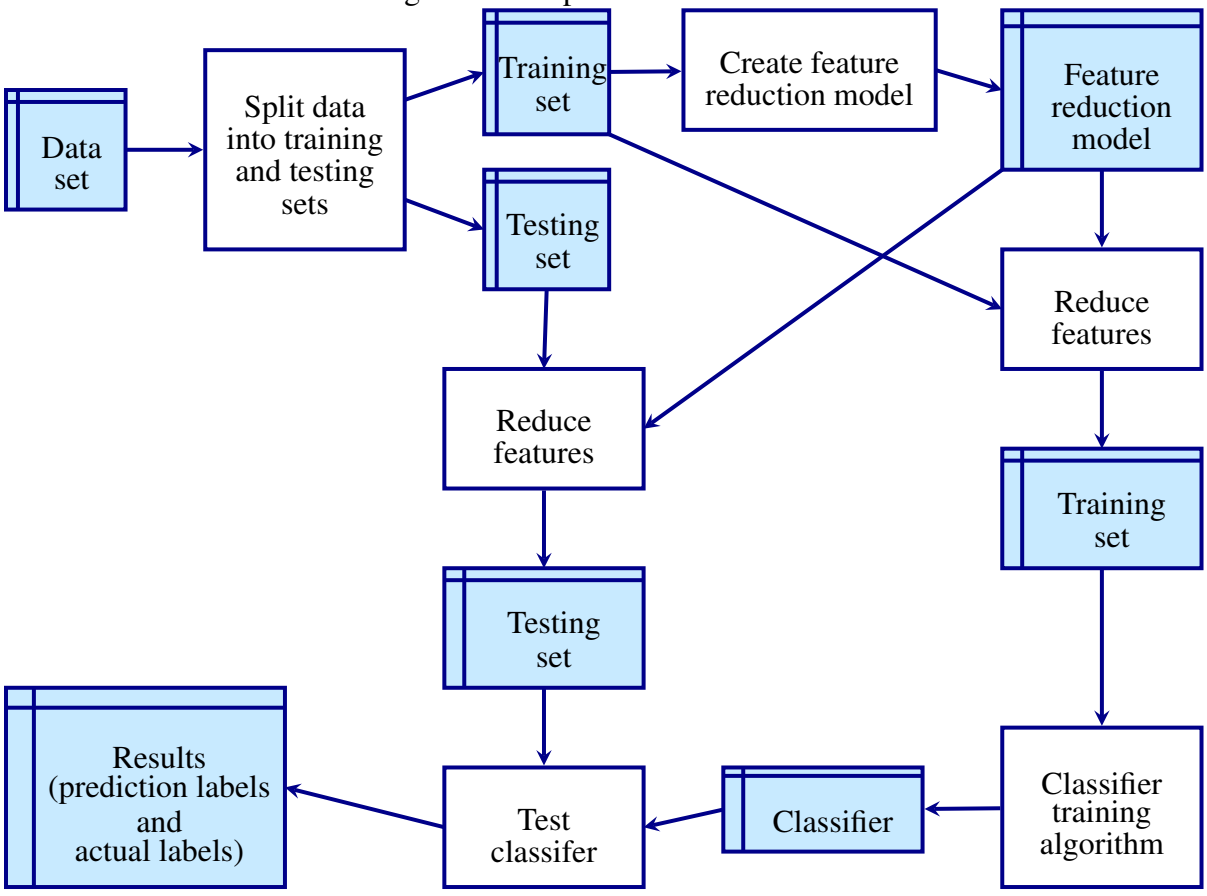


Figure 6.1 depicts the experiment process in graphical form. The blue boxes represent data structures while the white boxes represent processes. A data structure with an arrow

going into a process means the data structure is an input to the process. A data structure with an arrow coming out of a process means the data structure is an output of the process. The figure clearly reveals that only the training set is used as input to construct the feature reduction model and the classifier while the testing set is only used to test the classifier.

Completing the 180 experiments produces 180 result files. A result file contains the predicted strategy label and actual strategy label for each sample in the testing set. The following sections discuss how we process these results to reveal the knowledge hidden in the data—knowledge that answers the question *have the objectives been met?*

6.4 Classifier Accuracy

Objectives 2-5 are important as they allow us to objectively determine how to create the best classifier in terms of accuracy. The fist 3 objectives help us isolate each of the 3 dimensions of data set, feature reduction technique, and classifier to compare values within the dimension. Objective 5, on the other hand, looks at every combination of the 3 experiment dimensions to find the best overall combination. This combination can be used in future work to produce high-accuracy strategy classifiers for RTS game agents.

Objectives 2-5 must be validated. Objectives 2-4 are concerned with comparing the accuracy of our classifier across 3 of the dimensions of our experiments; data set, feature reduction technique, and classifier. To accomplish this, we compute the accuracy of each experiment. The accuracy is defined as the number of samples correctly predicted divided by the total number of samples in the experiment. The 180 experiments are segregated into 2 groups; those that used the Econ data set and those that used the No Econ data set. The mean of each group is computed. This process is repeated for the feature reduction techniques. The 180 experiment accuracies are segregated into 3 groups; None, PCA, and LDA. The mean for each group is computed. Finally, the 180 accuracies are segregated into K-NN and SVM and the mean is computed for each of the 2 groups. This gives us 7

values we can use to compare each dimension of our experiments: data set, feature reduction technique, and classifier.

Objective 5 requires us to find the combination of data set, feature reduction technique, and classifier that produces the classifier with the highest accuracy. To accomplish this, we take the same 180 experiment accuracies and split them into 12 groups according to the 12 combinations of data set, feature reduction technique, and classifier.

6.5 Classifier Execution Speed

Here we examine classifier execution speed in order to answer Objective 6: *Find the best combination of data set, feature reduction technique and classifier type in terms of speed of execution*. This objective is important because execution time is critical in RTS game simulations. By finding the experiment that produces the fastest executing classifier, future researchers can use this combination to produce high-speed strategy classifiers for RTS game agents.

To answer this objective, we examine the execution time elapsed in testing each of the 180 classifiers. These 180 time values are divided by the respective number of samples in each testing set to give 180 average, single-sample classification times. Each of the 12 combinations have 15 average, single-sample classification times (from 3 maps $\times$ 5 iterations). The 15 values of each combination are averaged. This gives us 12 final values which represent the mean time to classify a single sample. In addition, we also compute the standard deviation across the 15 values using the same method used to compute the mean.

6.6 Properties of Most Accurate Classifier and Fastest Executing Classifier

Objective 7 must also be dealt with. For Objective 7 we show by experiment that the most accurate classifier (revealed in Objective 5) and the fastest executing classifier (revealed in Objective 6) have the following 5 properties.

1. The standard deviation of the classifier's accuracy is under 5% of the mean of the classifier's accuracy.
2. The largest delta in accuracy per iteration is less than 5% for each of the 3 maps.
3. The producer accuracy is equal to or greater than 75% for every strategy.
4. The execution speed is under 10% of a game cycle period.
5. The accuracy is above 80% for over 90% of the time.

We must produce classifiers that are suitable for online RTS strategy classification. This is the motivation in examining these 2 classifiers to ensure the above properties. No matter how accurate a classifier is, if it can't execute in real-time, it is useless for our research. In the same way, no matter how quickly a classifier executes, if it has poor accuracy, it too is not fit to be used as a online RTS strategy classifier. These properties ensure that our classifiers meet a minimum accuracy bound and maximum execution speed bound to be considered acceptable classifiers for future research.

Property 1 ensures the classifier is precise. If the standard deviation is greater than 5%, the quality of the accuracy is questionable because the classifier is unable to consistently produce the same results. To show Property 1, the accuracy table from Objective 5 in the Classifier Accuracy Section is used.

Property 2, like Property 1, is a precision measure. It measures variance across the iterations of the 5-fold cross-validation. Hence, having more than 5% delta between

iterations calls into question the quality of the accuracy. For Property 2, we construct a table that gives the accuracy of the classifier for each fold of the 5-fold cross-validation.

To say with confidence that the classifier is accurate, one must ensure it is accurate in classifying any of the eight strategies. If a classifier has a high overall accuracy, but an accuracy below 75% for one or more strategy, the classifier cannot be declared accurate with confidence. Thus, the individual producer accuracies must be inspected to ensure none are below 75%. Property 3 is shown using a confusion matrix. We create a confusion matrix for each combination and each map. Thus, (12 combinations)·(3 maps) = 36 confusion matrices are created. Only the three confusion matrices relating to the most accurate classifier and the 3 matrices relating to the fastest executing classifier are presented in Chapter 7, the Results Chapter. All 36 matrices can be reviewed in detail in Appendix B. Each confusion matrix holds the results of all 5 iterations of 5-fold cross-validation.

Our agent must execute in soft real-time. Its deadline is the period of a game cycle. Since the game runs at 30 Hz, the period is 33.3 ms. However, the agent does not own all the compute resources, but must share them with the rest of the RTS game simulation. Thus, we restrict the classifier to only 10% of the compute resources, leaving the other 90% available for the game simulation. Hence, the classifier's deadline is 3.3 ms. Real-time concerns in RTS are discussed further in 2.5. We refer to the execution speed and standard deviation tables created for Objective 6 in the Classifier Execution Speed Section to validate Property 4.

Property 5 is intended to ensure the quality of the classifier accuracy over time. If the classifier is at least 80% accurate for over 90% of the time, we can have confidence that the accuracy of the classifier will be consistent for the duration of the game. The property is shown by plotting the average accuracy of the experiment for each of the 3 maps over

time. We create one such figure for each of the 12 combinations. These can be examined in Appendix B.

6.7 Chapter Summary

This chapter presents our 7 objectives and details our 5-fold cross-validation experiment process. We also discuss the importance of each objective and how we process the results of our experiments to answer the objectives.

The objectives are discussed along with the wins-losses and counter-strategies (Objective 1), the classifier accuracy (Objectives 2-5), the classifier execution speed (Objective 6), and finally, the properties of most accurate classifier and fastest executing classifier (Objective 7).

Objective 1 is intended to validate the strategy schema concept, the agent framework, and the strategy definitions in that they work together to produce agents that implement the intended strategies effectively.

Overall, Objectives 2-7 are chosen to validate our methodology produces strategy classifiers that are accurate and high speed—appropriate for use in online RTS strategy classification.

With the discussion on the design of our experiments complete, let us continue into our actual experiment results.

7 Results and Discussion

7.1 Introduction

In this chapter we present experimental results, explain how to interpret the tables and graphs, and discuss if and how our objectives have been met. The experiment objectives are enumerated and explained in Chapter 6. The remaining sections of this chapter cover the win-loss results, the counter-strategies, the Objective 1 requirements, the classifier accuracies, the classifier execution speeds, the properties of the most accurate classifier, the properties of the fastest executing classifier, and finally, the strategy observation fidelity. The Win-loss Results Section and the Counter-strategies Section along with the Objective 1 Requirements Section discuss Objective 1. The Classifier Accuracy Section deals with Objectives 2-5. The Classifier Execution Speed Section covers Objective 6. Objective 7 results are discussed in the next 2 sections, Properties of Most Accurate Classifier and Properties of the Fastest Executing Classifier.

7.2 Win-loss Results

Tables 7.1-7.3 show the 3 win-loss tables. One for each map: small, medium, and large. Positive values denote the number of wins while negative values denote the number of losses. The diagonals are all zeros because we did not test match-ups against like strategies. The upper and lower triangles give identical information—only inverted. We use the notation *cell (x, y)* to indicate the cell at row x, column y. The tables should be read from beginning of row to top of column.

For example, cell (1, 2), the second cell in the top row, reads IR lost 8 games against TR. Cell (2, 1) reads TR won 8 games against IR. Both cells contain the same information, only inverted because they are opposite in perspective. Since we repeat each match-up 20

Table 7.1: Number of Wins for each Match-up on the Small Map

	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle
IR	0	-8	12	6	20	18	-20	3
TR	8	0	2	10	18	20	2	6
Blitz	-12	-2	0	0	-4	10	16	16
Artil	-6	-10	0	0	6	18	8	10
Bomber	-20	-18	4	-6	0	-20	-4	-10
Antiair	-18	-20	-10	-18	20	0	-8	-12
Exp	20	-2	-16	-8	4	8	0	-14
Turtle	-3	-6	-16	-10	10	12	14	0

times, the values range from -20 to 20. A 20 means strategy A won all 20 games against strategy B; while a -20 means strategy A lost all 20 games against strategy B, and so forth.

Table 7.2: Number of Wins for each Match-up on the Medium Map

	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle
IR	0	-20	-20	-18	18	8	-19	-20
TR	20	0	-20	9	20	20	6	-17
Blitz	20	20	0	18	-14	-2	16	5
Artil	18	-9	-18	0	4	18	1	-17
Bomber	-18	-20	14	-4	0	-18	-15	-20
Antiair	-8	-20	2	-18	18	0	-15	-19
Exp	19	-6	-16	-1	15	15	0	-18
Turtle	20	17	-5	17	20	19	18	0

Table 7.3: Number of Wins for each Match-up on the Large Map

	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle
IR	0	-20	-20	-20	-6	-4	-20	-20
TR	20	0	-16	12	18	20	7	-17
Blitz	20	16	0	19	-14	-2	18	10
Artil	20	-12	-19	0	-8	2	0	-18
Bomber	6	-18	14	8	0	-20	-20	-20
Antiair	4	-20	2	-2	20	0	-17	-20
Exp	20	-7	-18	0	20	17	0	-19
Turtle	20	17	-10	18	20	20	19	0

Table 7.4: Number of Wins for each Match-up across all Maps

Strategy A	Strategy B	Wins on Small Map	Wins on Medium Map	Wins on Large Map	Strategy A	Strategy B	Wins on Small Map	Wins on Medium Map	Wins on Large Map
IR	TR	-8	-20	-20	Blitz	Bomber	-4	-14	-14
IR	Blitz	12	-20	-20	Blitz	Antiair	10	-2	-2
IR	Artil	6	-18	-20	Blitz	Exp	16	16	18
IR	Bomber	20	18	-6	Blitz	Turtle	16	5	10
IR	Antiair	18	8	-4	Artil	Bomber	6	4	-8
IR	Exp	-20	-19	-20	Artil	Antiair	18	18	2
IR	Turtle	3	-20	-20	Artil	Exp	8	1	0
TR	Blitz	2	-20	-16	Artil	Turtle	10	-17	-18
TR	Artil	10	9	12	Bomber	Antiair	-20	-18	-20
TR	Bomber	18	20	18	Bomber	Exp	-4	-15	-20
TR	Antiair	20	20	20	Bomber	Turtle	-10	-20	-20
TR	Exp	2	6	7	Antiair	Exp	-8	-15	-17
TR	Turtle	6	-17	-17	Antiair	Turtle	-12	-19	-20
Blitz	Artil	0	18	19	Exp	Turtle	-14	-18	-19

Table 7.4 presents the same data as the 3 win-loss tables. Here the information is aggregated into a single table to allow the reader to easily observe how a match-up behaves across the 3 maps. The notation is the same as with the win-loss tables. A positive number gives the number of wins of Strategy A over Strategy B; while a negative number gives the number of losses of Strategy A to Strategy B.

Table 7.5: Number of Winning Records Per Map

	Small	Medium	Large	Expected Affinity	Success
IR	5	2	0	Small	Yes
TR	7	5	5	Small	Yes
Blitz	3	5	5	Medium	Yes
Artil	4	4	2	Medium	Yes
Bomber	1	1	3	Large	Yes
Antiair	1	2	3	Large	Yes
Exp	3	3	3	N/A	N/A
Turtle	3	6	6	Large	Yes

In order to give a measure of each strategy's general performance per map, Table 7.5 is presented.

The values in the table represent the number of positive values that appear in the strategy's row for the given win-loss table according to map size. It is the number of winning records each strategy has in each map. So the larger the value, the better the strategy's performance on the given map. The success column refers to whether the strategy met the Objective 1 requirement of Table 6.1. For example, the Objective 1 requirement for IR is that it performs best on small maps. Clearly we see that IR has more winning records on the small map than on the large map, so it is considered a success. Exp does not have an Objective 1 requirement but is included in the table for completeness.

7.3 Counter-strategies

Table 7.6: Counter-strategies

	Best Counter-strategies		
Strategy	Small Map	Medium Map	Large Map
IR	Exp	TR Blitz Turtle	TR Blitz Artil Exp Turtle
TR	TR	Blitz	Turtle
Blitz	IR	Bomber	Bomber
Artil	TR	Blitz	Blitz
Bomber	IR Antiair	TR Turtle	Antiair Exp Turtle
Antiair	TR	TR	TR Turtle
Exp	Blitz	Turtle	Turtle
Turtle	Blitz	Blitz	Blitz

Using the data of the 3 win-loss tables, we record the most effective counter-strategy of each strategy on each map. The effectiveness of a counter-strategy is the number of times it wins over the given strategy. This information is shown in Table 7.6. If a strategy has multiple counter-strategies for a map, it means that all the listed counter-strategies scored the same while playing against the strategy. So, any of the listed counter-strategies are equally effective against the strategy on the given map.

7.4 Objective 1 Requirements

Table 7.7: Objective 1 Requirements

	Requirement	Met
1	No strategy dominates all other strategies on any map.	Yes
2	Every strategy has an overall winning record against at least one strategy on at least one map.	Yes
3	Every strategy is the best counter-strategy for at least one strategy on at least one map.	Partial
4	TR and IR favor small maps.	Yes
5	Blitz and Artil favor medium maps.	Yes
6	Bomber, Antiair, and Turtle favor large maps.	Yes
7	Antiair dominates Bomber.	Yes

Requirement 1. This requirement can be validated by examining the columns in Table 7.6—the Counter-strategies Table. If every entry in any column contains the same strategy, then that strategy dominates the map of the given column. No strategy dominates either of the 3 maps. For example, even though TR has a winning record against every strategy on the Small Map, it is not always the most effective strategy on the map. IR is a better choice against Blitz and Blitz is a better choice against Exp and Turtle. We conclude the requirement is fully met.

Requirement 2. This requirement can be validated by scanning the rows of the 3 win-loss maps. If any strategy lacks at least one positive value in its 3 rows (small, medium and large), then that strategy does not have any winning records. The only row laking a positive number is IR's row for the Large Map. However, IR has 7 winning records for the other 2 maps. Therefore all strategies have at least one winning record and the requirement is fully met.

Requirement 3. Although all 8 strategies appear as effective counter-strategies, neither Artil nor Antiair can claim to be the best counter-strategy as they always tie with at least one other strategy. Antiair is the best counter-strategy against Bomber for the small and

large maps, but shares that distinction with TR (Small Map) and Antiar, Exp, and Turtle (Large Map). However, Antiair consistently performs very well against Bomber for all 3 maps—taken as a whole, it performs better than any other strategy against Bomber. Therefore, it is still considered a valuable strategy. On the other hand, Artil, while very effective against IR on the Large Map, shares that distinction with 4 other strategies. Additionally, whenever Artil has a winning record, there is always another strategy that outperforms it. This makes Artil's value as a useful strategy questionable. Therefore, we conclude this requirement only partially fulfilled.

The reason for Artil's poor performance is that it does not effectively use its artillery units. Artil uses the same tactics as all other strategies (with exception to the Bomber strategy). It moves to the nearest enemy and attacks. To make Artil useful, the agent framework must be modified to have alternate tactics when artillery units are present in an attack group. Instead of attacking the nearest enemy, the attack group should target the closest enemy building. The artillery should be positioned on the outskirts of the enemy base, at their cannons' maximum range while still being able to hit the target building. Only the artillery should fire on the building while the escort tanks remain close to the artillery to defend them against enemy counter-attacks. Also, the attack group should retreat if the enemy's counter-attack is too strong. In this way the agent framework would better utilize the long rage of the artillery units and likely have a more effective Artil strategy.

Requirement 4. Both IR and TR have the most winning records on the Small Map, therefore this requirement is fully met.

Requirement 5. Both Blitz and Artil have their highest winning records on the Medium Map. However, Blitz ties on the Medium and Large Maps while Artil ties on the Small and Medium Maps. The ties do not contradict the objective, and overall, the data shows that they favor the Medium Map, therefore we conclude this requirement as met.

Requirement 6. Bomber, Antiair and Turtle have the highest winning record on the Large Map. This requirement is fulfilled.

Requirement 7. Antiair wins 20, 18, and 20 of 20 games against Bomber on the Small, Medium, and Large Maps respectively. Therefore, we can conclude that Antiair dominates Bomber on all maps—this requirement is fully met.

Exp Strategy Definition. Exp ties on all 3 maps and thus does not favor any map. One may have expected Exp to favor larger maps because it spends a large portion, 45%, of its resources on build power, economy, and defenses. Also, it builds a large army before attacking its enemy. These attributes normally benefit from a larger map. On the other hand, Exp only builds a small initial economy and then begins producing cheap infantry units. Therefore Exp also possesses attributes which favor small maps. Because of this mix, we left Exp out of the map size affinity requirement. However, Exp does perform according to its strategy definition.

Exp outperforms all other strategies against IR on the Small Map. Exp's success against IR on the Small Map is due to Exp constructing defenses and building light infantry units at the beginning of the game. Thus, when IR performs a rush attack, Exp is already prepared to counter it. Exp has winning records on all maps and is never the worst strategy on a given map. Therefore, it is a balanced strategy, which does not excel very often due to this balance—jack of all trades, master of none.

Verdict on Objective 1. Objective 1 is a very broad objective that is difficult to define completely. The only way for a reviewer to truly appreciate the dynamics of the strategy schema, agent framework, and strategy definitions is to sit down and watch the games being played in real-time (which is possible, since we have recorded all 1,680 games for future researchers to enjoy). We have done our best to make Objective 1 measurable by defining our seven requirements [Section 6.2]. Six of the seven requirements of Objective 1 are fully met. However, Requirement 3 is only partially met. Therefore, although, the

intent of Objective 1 has been fulfilled, it is not perfect. To really complete Objective 1 to the letter, the agent framework must be improved to better control attack groups with artillery units as discussed under Requirement 3.

7.5 Classifier Accuracy

Table 7.8: Accuracy Comparison Table

	Econ	No Econ	None	PCA	LDA	Knn	SVM
Mean	0.9405	0.9405	0.9409	0.9342	0.9464	0.9285	0.9525
Std Dev	0.0403	0.0399	0.0425	0.0450	0.0304	0.0425	0.0335

Table 7.8 provides the necessary information to answer Objectives 2-4. For the data set dimension, both Econ and No Econ perform nearly identically for the 180 experiments. In terms of feature reduction techniques, LDA has the highest overall mean accuracy with the lowest overall variance. SVM also has the highest mean accuracy and lowest variance when compared with K-NN.

So we find that LDA feature reduction technique and the SVM classifier produce the highest accuracy value while the choice of data set, in general, has little impact on accuracy performance.

Table 7.9: Overall Accuracy Means

	Econ			No Econ		
	None	PCA	LDA	None	PCA	LDA
Knn	0.926	0.912	0.943	0.929	0.920	0.941
SVM	0.956	0.955	0.951	0.953	0.949	0.950

Table 7.10: Overall Accuracy Standard Deviations

	Econ			No Econ		
	None	PCA	LDA	None	PCA	LDA
Knn	0.043	0.044	0.031	0.045	0.047	0.034
SVM	0.037	0.035	0.028	0.035	0.037	0.028

Now we turn our attention to Objective 5: *Find the best combination of data set, feature reduction technique and classifier type in terms of accuracy*. Tables 7.9 and 7.10 provide us with the information needed to answer this objective. Using the Econ data set with no feature reduction and SVM classifier produces the highest accuracy. Thus our highest accuracy classifier is defined by the experiment 3-tuple (Econ, None, SVM) and Objective 5 has been met.

Comparing Table 7.9 with Table 7.8 we see that they both agree that SVM produces more accurate classifiers than K-NN. However, using no feature reduction technique produces highest accuracies only for the SVM classifier. With K-NN, it is LDA that produces the highest accuracy. Additionally, it is interesting to note that the Econ data set tends to give higher accuracy when using the SVM classifier while the No Econ data set tends to give higher accuracy when using the K-NN classifier. However, this information is lost in Table 7.8 as it combines the SVM and K-NN classifers when examining the data sets, thus making the choice of data set appear to have no impact on the accuracy, when in fact it does.

Examining Table 7.10 reveals that SVM generally has least variance in its accuracies when compared with K-NN. Also LDA produces the lowest variance among the feature reduction techniques.

7.6 Classifier Execution Speed

Objective 6 asks us to find the combination that produces the fastest executing classifier. To answer the objective we look to Table 7.11. It is immediately evident that K-NN is much faster than SVN—by 2 orders of magnitude. This is expected as K-NN is merely computing the euclidean distance between the sample and its 56 labeled points. SVM, on the other hand, is using support vectors and Gaussian radial basis functions to determine in what class the samples falls. We see the fastest executing classifier is created from the combination of (No Econ, LDA, K-NN). Thus, Objective 6 is fulfilled.

Table 7.11: Mean of Execution Speed (µs)

	Econ			No Econ		
	None	PCA	LDA	None	PCA	LDA
Knn	8.61	7.46	7.15	8.22	7.39	7.14
SVM	926.40	703.77	898.21	827.31	686.74	762.59

Table 7.12: Standard Deviation of Execution Speed (µs)

	Econ			No Econ		
	None	PCA	LDA	None	PCA	LDA
Knn	0.05	0.08	0.13	0.12	0.13	0.10
SVM	310.86	224.94	247.83	259.28	233.04	176.18

Table 7.12 provides us with the standard deviation of the execution speeds. The standard deviation is always less than 2% of the mean for K-NN. However, not only is the SVM slower on overage, it also has large standard deviations. Therefore, the slowest executing classifier, (Econ, None, SVM), can have an execution speed of almost 2 ms at 3 standard deviations.

7.7 Properties of Most Accurate Classifier

Objective 5 revealed that the most accurate classifier is produced with the combination (Econ, None, SVM). We show that this classifier carries the 5 properties defined in Objective 7.

Property 1 requires *the standard deviation of the classifier's accuracy is under 5% of the mean of the classifier's accuracy*. Table 7.9 shows the mean accuracy is 0.956 and Table 7.10 shows the standard deviation is 0.037. Therefore, the ratio is 0.037/0.956 = 0.039. This is less than 0.05 so we have validated Property 1.

Property 2 is defined as *the largest delta in accuracy per iteration is less than 5% for each of the 3 maps*. This is shown in Table 7.13. The largest delta for the Small, Medium,

Table 7.13: Overall Accuracy for Experiment: Econ – None – SVM

Map Size	Iteration 1	2	3	4	5
Small	0.898	0.891	0.913	0.898	0.918
Medium	0.984	0.985	0.984	0.986	0.984
Large	0.981	0.981	0.977	0.979	0.976

and Large Maps is 0.027, 0.002, and 0.005 respectively—all well under 5%. Therefore, Property 2 also holds true.

Table 7.14: Confusion Matrix for Experiment: Econ – None – SVM on Small Map

	Predicted IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14627	0	0	9	57	0	108	0	14801	98.8%
TR	0	9530	21	210	145	0	0	23	9929	96.0%
Blitz	0	333	10401	66	855	1	0	317	11973	86.9%
Artil	0	505	502	8999	535	16	0	492	11049	81.4%
Bomber	0	12	71	158	9896	30	0	0	10167	97.3%
Antiair	0	3	62	154	2193	8024	0	0	10436	76.9%
Exp	169	0	4	46	2	39	15203	0	15463	98.3%
Turtle	0	399	1357	383	209	48	0	12576	14972	84.0%
Totals	14796	10782	12418	10025	13892	8158	15311	13408		
CA	98.9%	88.4%	83.8%	89.8%	71.2%	98.4%	99.3%	93.8%		90.3%

Tables 7.14, 7.15, and 7.16 present the 3 confusion matrices for the experiment. The far left column lists the actual strategies, while the top row list the predicted strategies. The second to last column is the total actual samples for each strategy. The second to last row is the total predicted samples for each strategy. The last column gives the producer accuracy (PA). PA is computed as the number of correctly predicted samples for a given strategy divided by the total number of actual samples for the given strategy. The bottom row gives the consumer accuracy (CA). CA is computed as the number of correctly predicted samples for given strategy divided by the total number of predicted samples for the given strategy. The diagonal gives the number of correctly predicted samples for each strategy. The upper and lower triangles give the counts of mis-predicted samples. For

example, looking at Table 7.14, the cell at (1, 5)—row 1, column 5—is read *of the 14,801 actual IR samples, the classifier mistakenly predicted 57 as Bomber*. This is a producer focus. Alternatively, for a consumer focus, the cell could be read as *of the 13,892 samples the classifier predicted to be Bomber, 57 where actually IR samples*. The number in the bottom corner is the overall accuracy—it is the sum of the diagonal divided by the sum of the entire matrix.

Table 7.15: Confusion Matrix for Experiment: Econ – None – SVM on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15619	0	0	0	0	0	0	0	15619	100.0%
TR	1	18217	0	0	4	0	0	16	18238	99.9%
Blitz	0	13	18335	7	446	0	0	44	18845	97.3%
Artil	0	8	24	24127	44	37	0	307	24547	98.3%
Bomber	0	26	0	0	11354	0	0	0	11380	99.8%
Antiair	0	2	0	0	1025	11556	0	0	12583	91.8%
Exp	105	0	0	1	14	24	25666	0	25810	99.4%
Turtle	0	0	95	72	39	2	0	25722	25930	99.2%
Totals	15725	18266	18454	24207	12926	11619	25666	26089		
CA	99.3%	99.7%	99.4%	99.7%	87.8%	99.5%	100.0%	98.6%		98.5%

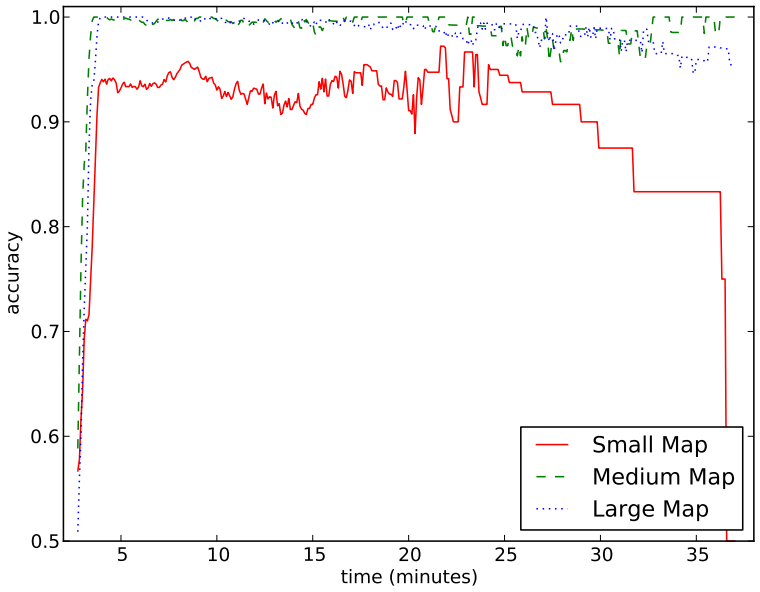
Table 7.16: Confusion Matrix for Experiment: Econ – None – SVM on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19434	24	0	0	0	0	1	0	19459	99.9%
TR	14	22601	0	0	51	0	0	138	22804	99.1%
Blitz	0	16	23188	9	686	0	0	36	23935	96.9%
Artil	7	18	86	28083	343	23	0	360	28920	97.1%
Bomber	21	5	81	0	13756	0	0	0	13863	99.2%
Antiair	32	0	84	0	1269	14874	0	0	16259	91.5%
Exp	31	26	2	23	19	16	34167	0	34284	99.7%
Turtle	0	7	3	313	302	1	0	29946	30572	98.0%
Totals	19539	22697	23444	28428	16426	14914	34168	30480		
CA	99.5%	99.6%	98.9%	98.8%	83.7%	99.7%	100.0%	98.2%		97.9%

Property 3 says *the producer accuracy is equal to or greater than 75% for every strategy*. To validate this we must examine the classifier's confusion matrices for each map. These are shown in Tables 7.14, 7.15 and 7.16. The far right column displays the producer accuracies. As it is shown, the lowest accuracy occurs in the Small Map with the Antiair strategy with a value of 76.9%. This is still above the 75% minimum; therefore, Property 3 holds.

Property 4 is true when *the execution speed is under 10% of a game cycle period.* A game cycle period is 33.3 ms; therefore, our classifier must execute in less than 3.3 ms. We use the worst-case execution speed within 3 standard deviations of the mean execution time as an upper bound for execution speed. This value is thus equal to or higher than 99.73% of execution speeds. Using Tables 7.11 and 7.12 we can compute this upper-bound execution speed. The mean execution speed is 0.9264 ms and one standard deviation is 0.31086 ms. Our upper-bound is $0.9264 + 3 \times 0.31086 = 1.859$ ms. So our upper-bound for 99.73% of the execution speeds is still under 3.3 ms; thus the classifier has Property 4.

Figure 7.1: Accuracy over Time for Experiment: Econ – None – SVM



The final property requires that *the accuracy is above 80% for over 90% of the time*. Let us examine Figure 7.1. The maximum game length is 37 minutes. However, the first 2 min 40 s of the game is not classified. Therefore, the game is only classified for 34 min 20 s. The graph shows all three accuracy curves stay above 80% for over 32, min. Thus, the accuracy curves are above 80% for 32/34.33 = 0.93 of the time. So the property holds.

Since all 5 properties hold, we have achieved half of Objective 7.

7.8 Properties of Fastest Executing Classifier

Objective 6 revealed that the fastest executing classifier is produced with the combination (No Econ, LDA, K-NN). We show that this classifier carries the 5 properties defined in Objective 7.

Property 1 requires *the standard deviation of the classifier's accuracy is under 5% of the mean of the classifier's accuracy*. Table 7.9 shows the mean accuracy is 0.941 and Table 7.10 shows the standard deviation is 0.034. Therefore, the ratio is 0.034/0.941 = 0.036. This is less than 0.05 so we have validated Property 1.

Table 7.17: Overall Accuracy for Experiment: No Econ – LDA – Knn

Map Size	Iteration 1	2	3	4	5
Small	0.892	0.870	0.916	0.892	0.913
Medium	0.961	0.965	0.956	0.974	0.969
Large	0.972	0.966	0.963	0.952	0.960

Property 2 is defined as *the largest delta in accuracy per iteration is less than 5% for each of the 3 maps*. This can be shown in Table 7.17. The largest delta for the Small, Medium, and Large Maps is 0.046, 0.018, and 0.02 respectively—all under 5%. Therefore, Property 2 also holds true.

Table 7.18: Confusion Matrix for Experiment: No Econ – LDA – Knn on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14673	0	11	0	20	0	97	0	14801	99.1%
TR	0	9699	70	101	27	0	0	32	9929	97.7%
Blitz	59	551	10051	282	552	2	0	476	11973	83.9%
Artil	28	768	264	9420	127	0	0	442	11049	85.3%
Bomber	104	87	331	37	8883	643	0	82	10167	87.4%
Antiair	105	62	374	88	1917	7832	0	58	10436	75.0%
Exp	395	9	0	5	0	0	15015	39	15463	97.1%
Turtle	58	831	461	553	60	60	0	12949	14972	86.5%
Totals	15422	12007	11562	10486	11586	8537	15112	14078		
CA	95.1%	80.8%	86.9%	89.8%	76.7%	91.7%	99.4%	92.0%		89.6%

Table 7.19: Confusion Matrix for Experiment: No Econ – LDA – Knn on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15594	0	0	0	0	0	25	0	15619	99.8%
TR	1	18179	0	0	0	0	22	36	18238	99.7%
Blitz	0	115	17891	153	353	1	0	332	18845	94.9%
Artil	3	232	262	23692	9	49	4	296	24547	96.5%
Bomber	0	3	90	0	10714	572	0	1	11380	94.1%
Antiair	10	16	91	5	706	11740	4	11	12583	93.3%
Exp	370	7	0	5	0	8	25415	5	25810	98.5%
Turtle	2	699	607	181	0	44	0	24397	25930	94.1%
Totals	15980	19251	18941	24036	11782	12414	25470	25078		
CA	97.6%	94.4%	94.5%	98.6%	90.9%	94.6%	99.8%	97.3%		96.5%

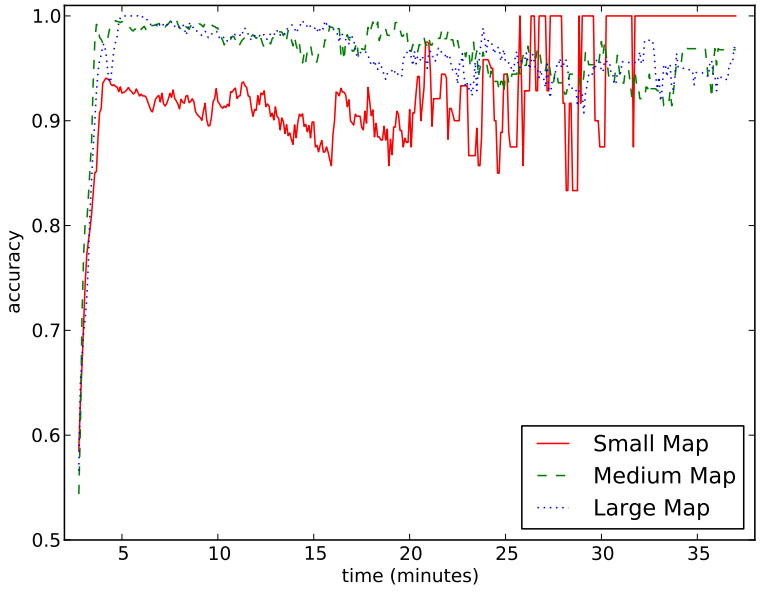
Property 3 says *the producer accuracy is equal to or greater than 75% for every strategy*. To validate this we must examine the classifier's confusion matrices for each map. These are shown in Tables 7.18, 7.19 and 7.20. The far right column displays the producer accuracies. As it is shown, the lowest accuracy occurs in the Small Map with the Antiair strategy with a value of 75.0%. This is equal to the 75% minimum; therefore, Property 3 holds.

Property 4 is true when *the execution speed is under 10% of a game cycle period.* A game cycle period is 33.3 ms; therefore, our classifier must execute in less than 3.3 ms.

Table 7.20: Confusion Matrix for Experiment: No Econ – LDA – Knn on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19430	26	0	0	0	0	3	0	19459	99.9%
TR	10	22773	0	21	0	0	0	0	22804	99.9%
Blitz	1	139	22801	14	273	1	0	706	23935	95.3%
Artil	40	269	210	27468	22	11	3	897	28920	95.0%
Bomber	27	20	410	0	13108	202	0	96	13863	94.6%
Antiair	7	7	421	1	890	14826	12	95	16259	91.2%
Exp	179	27	5	6	0	0	34042	25	34284	99.3%
Turtle	6	555	968	450	0	2	4	28587	30572	93.5%
Totals	19700	23816	24815	27960	14293	15042	34064	30406		
CA	98.6%	95.6%	91.9%	98.2%	91.7%	98.6%	99.9%	94.0%		96.3%

Figure 7.2: Accuracy over Time for Experiment: No Econ – LDA – Knn



We use the worst-case execution speed within 3 standard deviations of the mean execution time as an upper bound for execution speed. This value is thus equal to or higher than 99.73% of execution speeds. Using Tables 7.11 and 7.12 we can compute this upper-bound execution speed. The mean execution speed is 0.00714 ms and one standard deviation is 0.0001 ms. Our upper-bound is $0.00714 + 3 \times 0.0001 = 0.00744$ ms. So our

upper-bound for 99.73% of the execution speeds is still well under 3.3 ms; thus the classifier has Property 4.

The final property requires that *the accuracy is above 80% for over 90% of the time*. Let us examine Figure 7.2. The maximum game length is 37 minutes. However, the first 2 min 40 s of the game is not classified. Therefore, the game is only classified for 34 min 20 s. The graph shows all three accuracy curves stay above 80% for over 33, min. Thus, the accuracy curves are above 80% for 33/34.33 = 0.96 of the time. So the property holds.

Since all 5 properties hold, we have achieved the other half of Objective 7.

7.9 Strategy Observation Fidelity

It is important to consider these results are for a 5 second period. In other words, using any of these classifiers, the agent could predict the opponent's strategy once every 5 seconds with the above accuracies. However, this strategy fidelity is much higher than is actually needed in practice. In reality, a player does not change strategy every 5 seconds–not even every minute. A player may only change strategies a handful of times during the entire game—or if the initial strategy works well, the player may not change strategies at all. It takes time to actually transition from one strategy to another. Changing strategies likely involves building units of different types and/or constructing new buildings. Thus, changing strategies too often results in wasted resources, since the player invests in one strategy but never gets to see any return on the investment because the player changes to the next strategy before fully applying the previous strategy. The player never makes any real progress towards the goal of winning the game because all the player's resources are wasted transitioning between strategies.

The agent really doesn't need to predict the opponent's strategy every 5 seconds. Predicting the strategy every minute is fast enough since the opponent is not be able to change strategies successfully more than once per minute. Therefore, the effective accuracy of the agent can be much higher if instead of using a single prediction every 5

seconds, it uses the 12 predictions in the last minute (5 * 12 = 60) to create a set of 12 votes. The agent then takes the highest voted strategy as the predicted strategy. The general equation for determining the effective accuracy is given by Equation 7.1 where n is the total number of votes and p is the producer accuracy for the given strategy. In other words p is the probability of a vote being correct. Of course, this equation assumes the opponent does not change his strategy during the course of collecting the n sample predictions.

$$\text{EffectiveAccuracy}(n, p) = \sum_{k=n/2+1}^{n} \binom{n}{k} p^k (1-p)^{n-k} \tag{7.1}$$

$$\text{probability(k, n)} = \binom{n}{k} p^k (1-p)^{n-k} \tag{7.2}$$

This equation actually gives a lower-bound, worst-case accuracy. We assume that if any of the votes are wrong, they all predict the same wrong strategy. In other words, it is the probability that the agent receives at least n/2 + 1 correct votes. This is the same equation one would use to determine the probability of flipping heads at least n/2 + 1 times using a biased coin with probability of heads = p. It is based on Equation 7.2 which gives the probability of exactly k successes in n independent Bernoulli trials, with probability of success p and probability of failure $1 - p$ as defined in Section 6.2 on page 407 of [33].

Table 7.21: Effective Accuracy for 1 min Fidelity

	Producer Accuracy for a Given Strategy (p)					
	0.70	0.75	0.80	0.85	0.90	0.95
6	0.74	0.83	0.90	0.95	0.98	0.997
12	0.88	0.95	0.98	0.995	0.9995	0.99999
18	0.94	0.98	0.996	0.9995	0.99998	0.9999999

Table 7.21 gives the effective accuracies for p = 0.70 to 0.95 in increments of 0.05 and for n = 6, 12 and 18. Let us consider the experiment using the data set without economy features, LDA feature reduction, and the K-NN classifier on the small map. The

confusion matrix is shown in Table 7.18. The lowest producer accuracy occurs with the Antiair strategy at 0.75. Using the recommended 12 sample predictions per minute, even the worst case producer accuracy, our effective accuracy would be 0.88—a 13% increase.

7.10 Chapter Summary

We have presented our results and discussed our objectives for the win-loss results, counter-strategies, classifier accuracy, classifier execution speed, the properties of the most accurate classifier, and lastly, the properties of the fastest executing classifier. Additionally, we have explored the implications of the strategy observation fidelity.

Of our 7 objectives, Objectives 2-7 are fully satisfied. Only Objective 1 is a partial success. Although our agent framework largely meets the intent of Objective 1, there is still some room for improvement. Specifically, the agent framework tactics should be improved when artillery units are present in an attack group to better utilize the strengths and capabilities of the aforementioned artillery units so that strategy definitions which leverage artillery units will be more effective. This is discussed in Section 7.4. As it is, including artillery units in one's strategy definition is likely to be a poor choice as the artillery units will not be used to their full potential. However, creating a strategy definition without the artillery units can be very effective as we have shown in Sections 7.2 through 7.4 with our other seven strategy definitions.

The presentation of experiment results is complete and we now move into the conclusion and discussion of future work.

8 Conclusion and Future Work

8.1 Conclusion

This research effort is concerned with the advancement of computer generated forces AI for Department of Defense (DoD) military training and education. The vision of this work is agents capable of perceiving and intelligently responding to opponent strategies in real-time. The research goal is to lay the foundations for such an agent using the Balanced Annihilation real-time strategy (RTS) game running on the Spring engine. We defined six RTS AI research objectives in Chapter 1. The first four research objectives are 1) create a strategy definition schema, 2) formulate eight strategy definitions, 3) design and implement an agent framework, and 4) generate a data set. These four research objectives are evaluated with Experiment Objective 1 [Sections 6.2, 7.4]. Research Objective 5, create an accurate, real-time strategy classifier, is evaluated with experiment objectives 2-7 [Sections 6.4-6.6, 7.5-7.8]. Research Objective 6, build a counter-strategy table, is also evaluated with Experiment Objective 1 [Sections 6.2, 7.4]. The following paragraphs address the success of each research objective.

Strategy Definition Schema. The strategy definition schema concept, defined in Section 4.2.1, is very successful as it was the driving vision behind the agent architecture and shaped it into a very flexible design able to accommodate a wide range of strategies. Typically, researchers in RTS AI create hand-scripted agents to run experiments [Chapter 2]. If future researchers want to expand on the work by using more agents with new behavior, they have to hand-code each agent—a tedious and time consuming task. On the other hand, our work has produced an agent framework that can provide thousands of agents by simply varying the strategy definition. Future researchers can harness the flexibility of the schema to quickly and easily create new agent behavior by writing simple strategy definitions. However, the schema cannot be evaluated in isolation. It is the

conceptual groundwork on top of which the strategy definitions and the agent framework are built. To assess the schema's value, the schema, strategy definitions, and framework must be examined as a whole. That is why Experiment Objective 1 serves as an evaluation for the first four research objectives.

Eight Strategy Definitions. The strategy definitions are presented in Table 4.3 of Section 4.2.11. This research objective is to formulate 8 balanced strategy definitions. Experiment Objective 1 is intended to validate the balanced property of our strategy definitions—specifically, the first 3 requirements of Experiment Objective 1 found in Table 6.1 of Section 6.2. By meeting these requirements, we show that each strategy is useful and has value while no one strategy dominates all others. We also show that our strategy schema is able to represent a wide range of strategies. However, like the strategy definition schema, the strategy definitions are useless on there own—merely concepts and ideas. It is these ideas together with the implemented agent framework that gives us a something measurable!

Agent Framework. The agent framework is designed with the capability of playing any strategy definable with our strategy definition schema. The concept of the strategy definition schema drove the agent design decisions. The agent framework is the realization of the schema concept and the implementation engine of the strategy definitions. One area should be improved to fully meet Requirement 3 [Sections 6.2, 7.4]. The issue is exposed in the Artil strategy. The agent framework tactics when controlling attack groups with artillery present must be improved to leverage the artillery units' strengths and negate its weaknesses. However, the experiments validate the agent can control the other 13 units correctly and is able to play the other seven strategies effectively. Aside from the minor issue of artillery control, all the requirements are met and Experiment Objective 1 is fulfilled.

There is still one more property that should be discussed before claiming success over Research Objective 3. Our research objective requires an agent framework that executes in real-time. The real-time property of the agent framework is validated in Section 5.2 by Table 5.2. The "*Speedup*" row gives the average speedup of the games executed during data collection. The entire game simulation, in addition to running 2 of our agents, is able to execute approximately 21 to 31 times faster that normal game speed. This means the agents, on average, are able to compute over 20 game cycles in the time allotted for only one game cycle. Although this does not prove the agent can meet a hard real-time requirement, it does validate the agent framework's ability to meet the soft real-time requirement—that is, the mean time for the agent to executes a cycle is much lower than the time allotted for a game cycle!

Thus, the agent framework is both real-time and effective in playing our 8 strategy definitions. In other words, we have successfully fulfilled Research Objective 3.

Data Set. Our data collection process is explained in Chapter 5: Real-time Strategy Classification. We have collect 1,680 games, on 3 different maps, using 8 strategies. This provides a large, varied data set. The game observations include unit positions, which were not used but may be of interest to other areas of RTS research. We have also recorded all 1,680 game replays and command line outputs, which allows future researchers to examine exactly what transpired in each and every game. Researchers can validate our data set and have confidence knowing what they are using. Because of the size, breadth, and transparency of our data set, and our success in using it for training strategy classifiers and finding counter-strategies, we conclude Research Objective 4 is fulfilled.

Strategy Classifier. The strategy classifier builds on the products of the previous four research objectives. Hence it has six experiment objectives to meet. Because validating the classifier also validates the data set, which in turn validates the schema, definitions, and framework, most of our testing effort revolved around the strategy classifiers. We

found that using the (Econ, None, SVM) combination produced the most accurate classifier; while using the (No Econ, LDA, K-NN) combination produced the fastest executing classifier. Both classifiers have accuracies that exceed the 85% minimum and execution times below the 3 ms maximum. Therefore, we have successfully met Research Objective 5.

Counter-Strategy Table. The counter-strategy table is trivial to construct as it merely builds on the successes of research objectives 1-4. Creating well balanced strategy definitions and an agent framework that can effectively play the strategies produced a research platform we could use for data collection. With the data set in hand, finding counter-strategies is as simple as counting wins and losses for each match-up on each map.

Through this research effort, we have provided the RTS AI field with six contributions available for future research in strategy-focused RTS AI which will benefit both military and academic institutions:

1. Strategy definition schema effective in defining a range of RTS strategies.
2. Seven strategy definitions
3. Real-time agent framework able to play any strategy expressible by our strategy definition schema
4. Strategy-focused, RTS game data set
5. Accurate, real-time strategy classifier
6. Effective counter-strategy table

These contributions lay a foundation for a truly dynamic real-time RTS agent able to perceive and adapt to opponent strategies. And this foundation is exactly the goal we were aiming to achieve!

8.2 Future Work

8.2.1 Dynamic Strategy Agent. As revealed in the Introduction Chapter, our vision is to create an agent that can perceive and respond to opponent strategy. The next logical step in our research is to use the contributions of this thesis work—the agent framework, classifier and counter-strategy table—to create a dynamic strategy agent. The agent framework must be modified as follows. It should periodically (we suggest every 5 seconds) collect the opponent's game state and use our classifier to predict the opponent's strategy. Then, over a larger period of time (we suggest every minute), use the n most recent predictions as strategy votes. The strategy with the most votes wins. Then the agent should use the counter-strategy table to lookup an effective strategy to counter the predicted opponent's strategy with. In addition, the agent should construct economy structures (metal extractors an solar panels) and defensive structures (light laser towers and anti-air turrets) whenever it is unsure of the opponents strategy. This would be the case for approximately the first 2 min 40 s of the game. The modifications to the agent framework are minimal—just a few dozen lines of code. But the effect is a first generation dynamic strategy agent for RTS games.

8.2.2 Design of Experiments. Once a dynamic strategy agent is constructed, the dynamic strategy concept must be validated. Suggestions for testing a dynamic strategy agent follow:

Static Strategy Agents. Test the modified agent framework against the original 8 strategy definitions. In other words, the opponent is given a basic strategy to implement and does not deviate from the given strategy for the duration of the game. The motivation for this testing is to show the dynamic strategy agent can recognize the basic strategies and respond with appropriate, simple, counter-strategies. The agent is expected to performs very well since these are the same 8 strategies used to train the classifier and counter-strategy table.

Scripted Strategy Agents. Test our dynamic strategy agent against agents with changing strategy definitions. The opponent has a script which dictates what strategy definitions are to be used and when to implement them. The motivation for this testing is to show the agent can dynamically detect changes in basic, opponent strategies and respond with appropriate, simple counter-strategies.

Random Agent. Modify the agent framework to include a "*random*" mode where the agent randomly selects a strategy from the 8 strategy definitions every minute, and plays that strategy. The random agent has the exact same internal structure as our dynamic strategy agent; however, it replaces the classifier and counter-strategy table with a component that chooses a random counter-strategy at each cycle. Test the random agent against the static strategy agents and scripted strategy agents. Also, play the dynamic strategy agent against the random agent directly. Compare the performance of our dynamic strategy agent with the random agent. This experiment provides a ground truth to determine what performance improvements from the dynamic strategy agent are due to the novel AI algorithms and data structures (classifier and counter-strategy selection) used to make strategic decisions as opposed to the tactics built into agent framework.

8.2.3 Counter-Strategies. In order to create the counter-strategy table, we limited the search space to the 8 original strategy definitions. However, this restriction is not necessary. Future research could involve searching for optimal counter-strategies for each of the 8 strategies on each map. This could greatly improve the performance of the dynamic strategy agent as well as lead to the automated discovery of "*good*" strategy definitions with little or not human intervention required.

One search approach could use an evolutionary algorithm to generate and search strategy definitions to find an "optimal" counter-strategy for each strategy. Strategy variables become the parameters of integer-valued chromosomes.

The fitness function can be computed by running a number of games against the agent using a strategy definition in the population. This is an expensive fitness function, requiring approximately 20 to 45 seconds, on average, to compute if the game is run at maximum speed. However, the algorithm is easily parallelizable, since each game can be executed on a separate machine.

8.2.4 Other Thoughts on Future Work.

- Integrate a scouting module into the framework to allow for operation in imperfect information environments (i.e. fog of war)
- Integrate tactical-level contributions from recent RTS AI research [Chapter 2] into the framework
- Expand number of strategies.
- Build a simulator for faster learning. This will allow a more practical implementation of a genetic algorithm to find optimal counter-strategy definitions.
- Test against human opponents.
- Port approach to real-word Air Force wargames used for educating and training future strategic thinkers.
- Port approach to the Stracraft Brood War API [18] and test in annual Starcraft AI competitions [10, 24].

8.3 Final Remarks

With exception of the artillery units, all the objectives of the research have been achieved and the goal successfully met. We have laid the foundations for a dynamic strategy agent with our agent framework and strategy classifier. These are large pieces of the puzzle; the vision of creating dynamic strategy agents for RTS games is now much

closer to reality. Once the dynamic strategy agent concept is validated, these methodologies and techniques can be ported to computer generated forces of next-generation DoD wargames to enhance military training and education.

Appendix A: Sequence Diagrams

For each use case presented in Figure 4.15, a sequence diagram is depicted below in Figures A.1 trough A.10. As with the use case diagram, the Spring Engine is rendered as an actor outside the Agent system. The objects in the sequence diagrams are labeled by Class name. The Class names can be cross-referenced with Figures 4.12, 4.13, and 4.14.

Figure A.1: Update Sequence Diagram

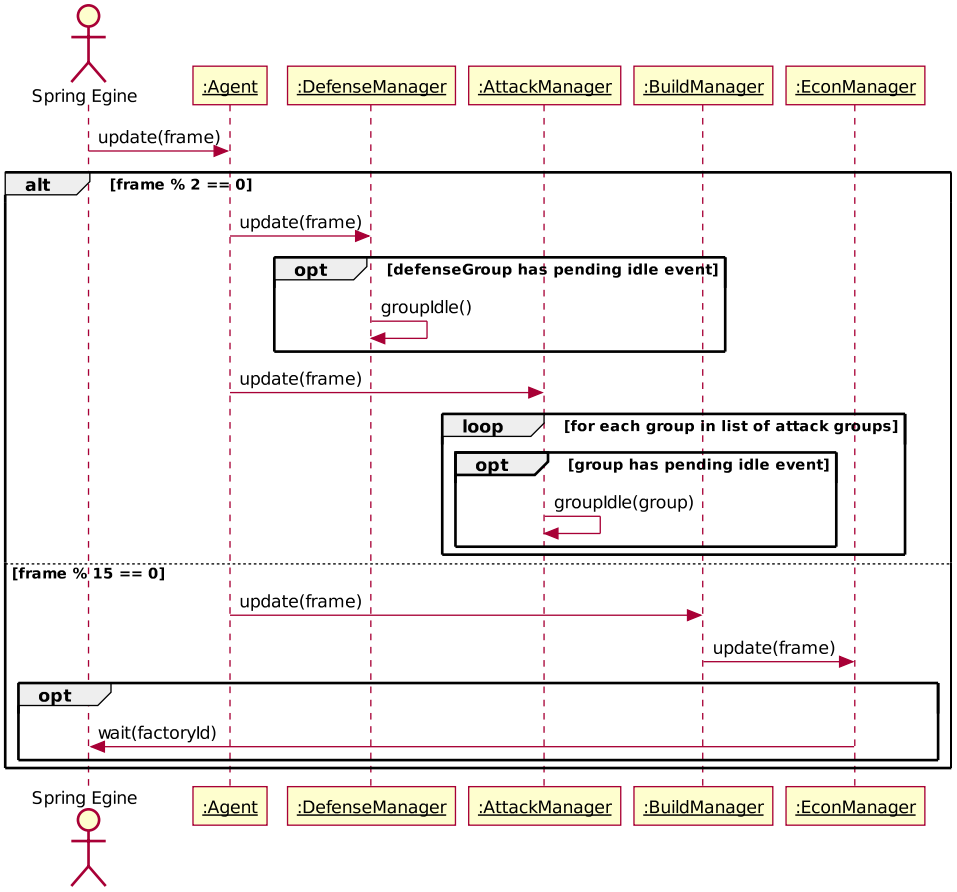


Figure A.2: Unit Created Sequence Diagram

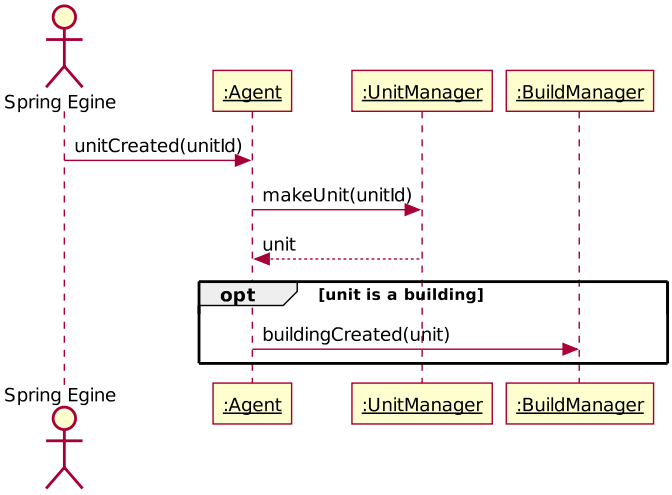


Figure A.3: Unit Finished Sequence Diagram

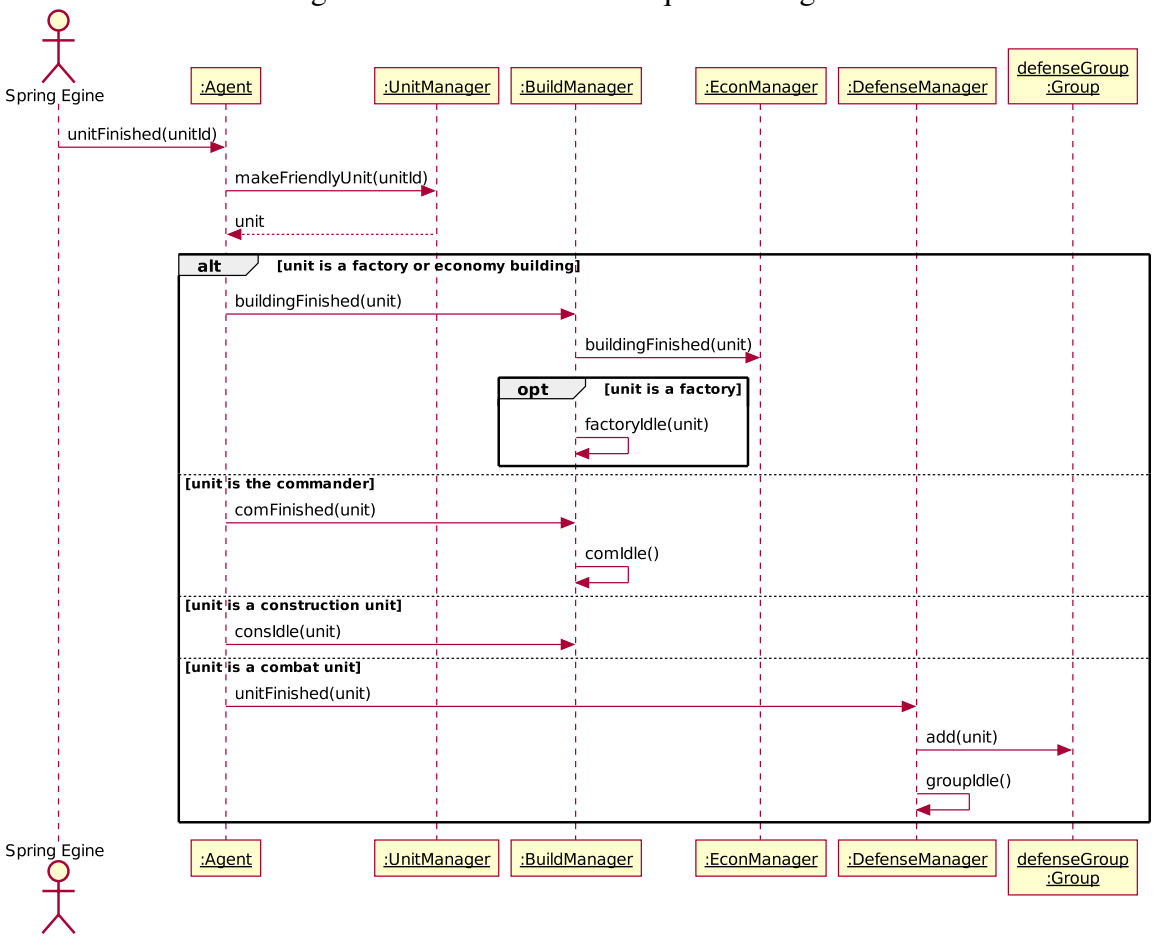


Figure A.4: Unit Idle Sequence Diagram

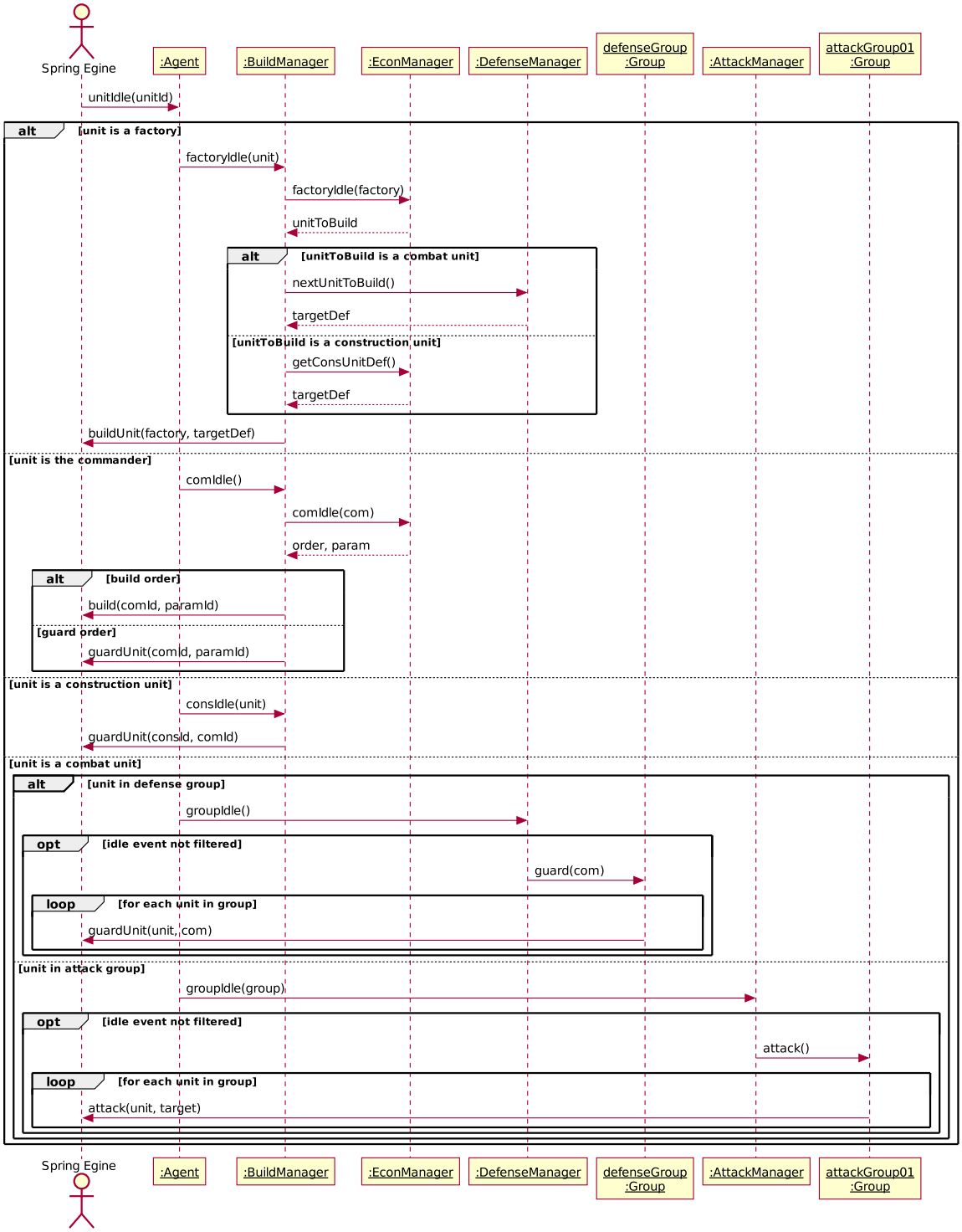


Figure A.5: Unit Damaged Sequence Diagram

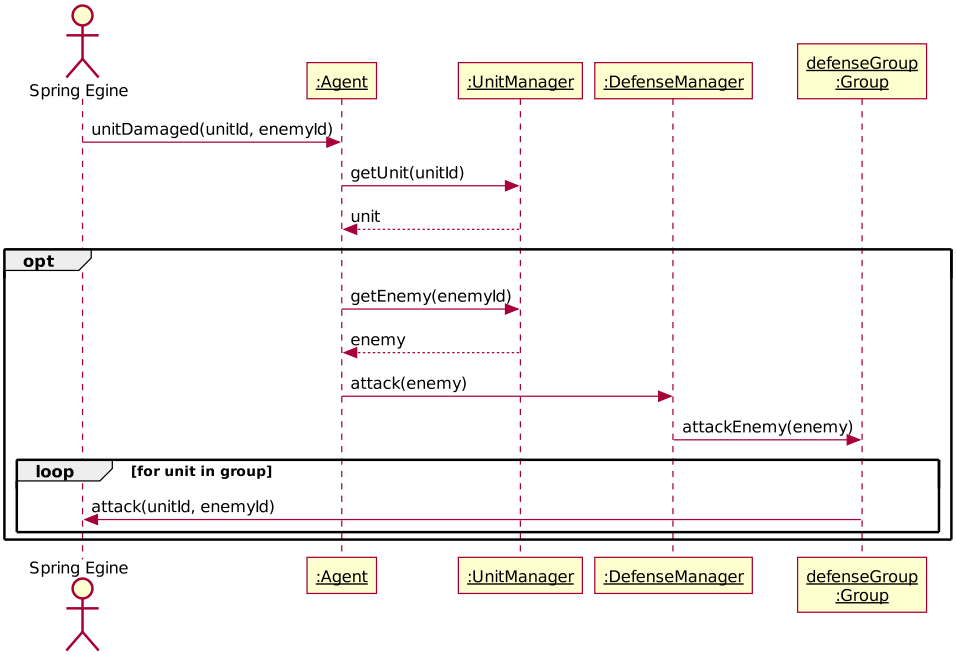


Figure A.6: Unit Destroyed Sequence Diagram

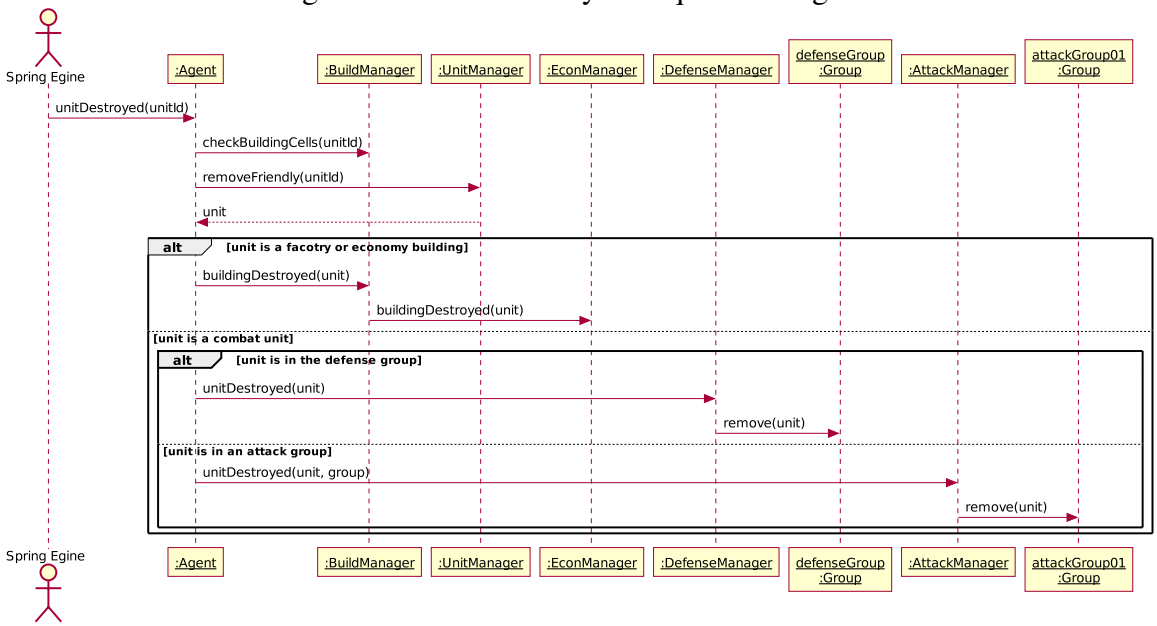


Figure A.7: Enter Line of Sight Sequence Diagram

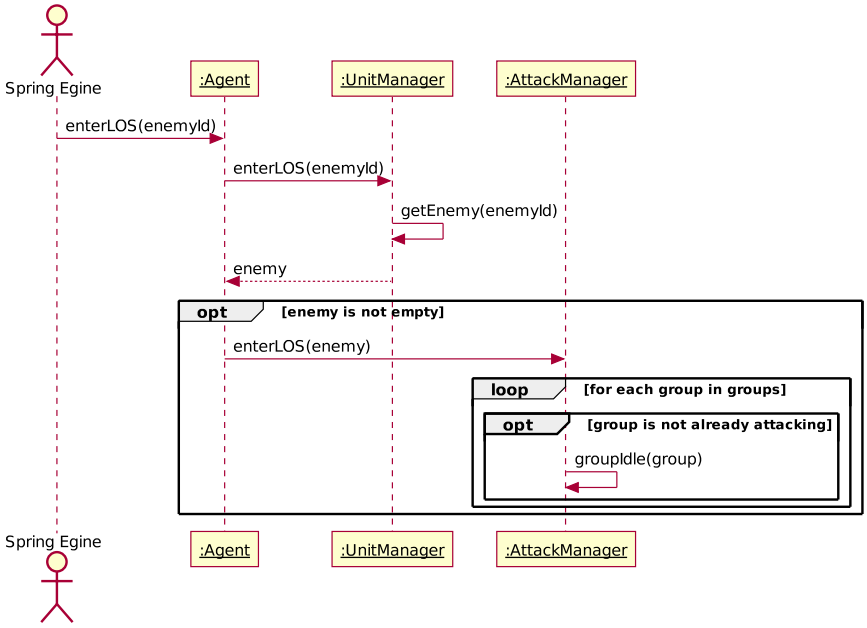


Figure A.8: Leave Line of Sight Sequence Diagram

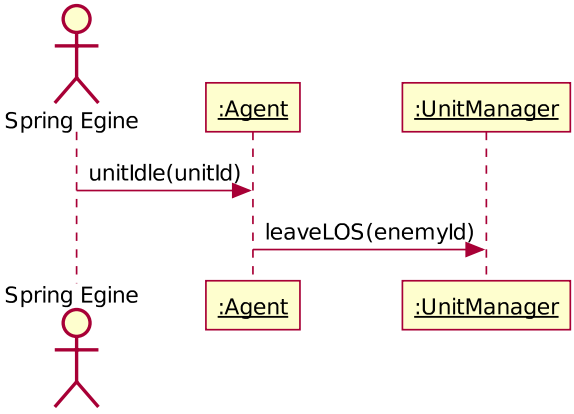


Figure A.9: Enemy Destroyed Sequence Diagram

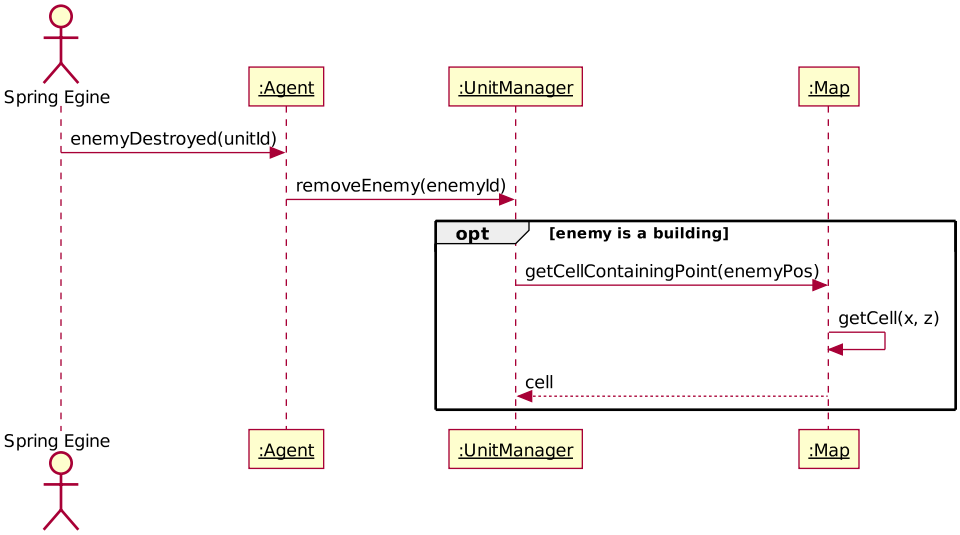
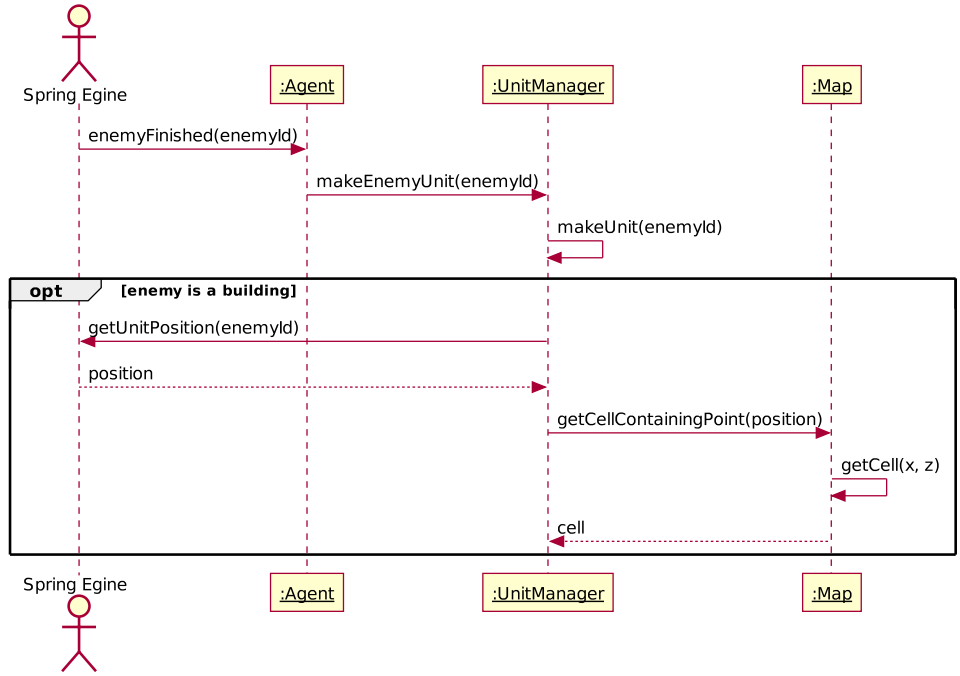


Figure A.10: Enemy Finished Sequence Diagram



Appendix B: Complete Experiment Results

B.1 Win–loss

Tables B.1, B.2 and B.3 provide the number of wins, ties, and losses for each match-up among the eight strategies for the small, medium and large maps respectively. Each strategy in the first column is player 1 and the each strategy in the top row is player 2. So the cell in row 2, column 1 is read as tank rush (TR) winning against infantry rush (IR) 9 times, tying infantry rush 0 times and loosing to infantry rush once.

Table B.1: Number of Wins, Ties, and Losses for each Match-up on the Small Map

	IR			TR			Blitz			Artil			Bomber			Antiair			Exp			Turtle		
	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
IR				5	0	5	6	0	4	7	0	3	10	0	0	10	0	0	0	0	10	4	1	5
TR	9	0	1				1	0	9	5	0	5	9	0	1	10	0	0	4	0	6	7	0	3
Blitz	0	0	10	0	0	10				5	0	5	4	0	6	6	0	4	9	0	1	8	0	2
Artil	4	0	6	0	0	10	5	0	5				5	0	5	10	0	0	7	0	3	7	0	3
Bomber	0	0	10	0	0	10	6	0	4	2	0	8				0	0	10	3	0	7	2	0	8
Antiair	1	0	9	0	0	10	1	0	9	1	0	9	10	0	0				2	0	8	2	0	8
Exp	10	0	0	3	0	7	1	0	9	3	0	7	5	0	5	6	0	4				1	0	9
Turtle	3	0	7	4	0	6	0	0	10	2	0	8	7	0	3	8	0	2	8	0	2			

Table B.2: Number of Wins, Ties, and Losses for each Match-up on the Medium Map

	IR			TR			Blitz			Artil			Bomber			Antiair			Exp			Turtle		
	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
IR				0	0	10	0	0	10	1	0	9	9	0	1	6	0	4	0	0	10	0	0	10
TR	10	0	0				0	0	10	6	4	0	10	0	0	10	0	0	7	0	3	1	1	8
Blitz	10	0	0	10	0	0				10	0	0	2	0	8	3	0	7	10	0	0	5	4	1
Artil	10	0	0	0	7	3	1	0	9				8	0	2	10	0	0	3	5	2	0	0	10
Bomber	0	0	10	0	0	10	9	0	1	6	0	4				1	0	9	2	1	7	0	0	10
Antiair	2	0	8	0	0	10	4	0	6	1	0	9	10	0	0				2	1	7	0	1	9
Exp	9	1	0	4	0	6	2	0	8	3	4	3	10	0	0	10	0	0				0	0	10
Turtle	10	0	0	10	0	0	4	1	5	8	1	1	10	0	0	10	0	0	9	0	1			

Table B.3: Number of Wins, Ties, and Losses for each Match-up on the Large Map

	IR			TR			Blitz			Artil			Bomber			Antiair			Exp			Turtle		
	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
IR				0	0	10	0	0	10	0	0	10	1	0	9	4	0	6	0	0	10	0	0	10
TR	10	0	0				1	0	9	6	4	0	9	0	1	10	0	0	4	3	3	1	1	8
Blitz	10	0	0	9	0	1				9	1	0	2	0	8	4	0	6	10	0	0	6	1	3
Artil	10	0	0	0	4	6	0	0	10				2	0	8	5	0	5	4	5	1	0	2	8
Bomber	4	0	6	0	0	10	9	0	1	6	0	4				0	0	10	0	0	10	0	0	10
Antiair	6	0	4	0	0	10	5	0	5	4	0	6	10	0	0				0	1	9	0	0	10
Exp	10	0	0	0	4	6	1	0	9	4	5	1	10	0	0	9	0	1				0	1	9
Turtle	10	0	0	10	0	0	1	1	8	10	0	0	10	0	0	10	0	0	10	0	0			

B.2 Overall Accuracy

Tables B.4 through B.15 show the overall accuracy of the experiments for the 12 combinations of data set, feature reduction technique, and classifier. The accuracies are given for each iteration and map size.

Table B.4: Overall Accuracy for Experiment: Econ – None – Knn

Map Size	Iteration 1	2	3	4	5
Small	0.856	0.848	0.889	0.870	0.869
Medium	0.944	0.949	0.950	0.965	0.960
Large	0.962	0.961	0.954	0.964	0.952

Table B.5: Overall Accuracy for Experiment: Econ – None – SVM

Map Size	Iteration 1	2	3	4	5
Small	0.898	0.891	0.913	0.898	0.918
Medium	0.984	0.985	0.984	0.986	0.984
Large	0.981	0.981	0.977	0.979	0.976

Table B.6: Overall Accuracy for Experiment: Econ – PCA – Knn

Map Size	Iteration 1	2	3	4	5
Small	0.837	0.829	0.865	0.858	0.872
Medium	0.927	0.952	0.927	0.962	0.946
Large	0.944	0.937	0.935	0.944	0.938

Table B.7: Overall Accuracy for Experiment: Econ – PCA – SVM

Map Size	Iteration 1	2	3	4	5
Small	0.893	0.892	0.914	0.918	0.921
Medium	0.980	0.983	0.983	0.986	0.986
Large	0.977	0.977	0.976	0.972	0.973

Table B.8: Overall Accuracy for Experiment: Econ – LDA – Knn

Map Size	Iteration 1	2	3	4	5
Small	0.901	0.875	0.908	0.905	0.919
Medium	0.963	0.966	0.969	0.969	0.970
Large	0.962	0.964	0.960	0.957	0.957

Table B.9: Overall Accuracy for Experiment: Econ – LDA – SVM

Map Size	Iteration 1	2	3	4	5
Small	0.913	0.888	0.934	0.912	0.930
Medium	0.954	0.979	0.968	0.973	0.976
Large	0.975	0.973	0.966	0.966	0.963

Table B.10: Overall Accuracy for Experiment: No Econ – None – Knn

Map Size	Iteration 1	2	3	4	5
Small	0.839	0.857	0.898	0.858	0.892
Medium	0.960	0.965	0.947	0.967	0.971
Large	0.960	0.964	0.944	0.952	0.956

Table B.11: Overall Accuracy for Experiment: No Econ – None – SVM

Map Size	Iteration 1	2	3	4	5
Small	0.896	0.889	0.910	0.913	0.916
Medium	0.978	0.978	0.973	0.977	0.978
Large	0.980	0.979	0.977	0.977	0.974

Table B.12: Overall Accuracy for Experiment: No Econ – PCA – Knn

Map Size	Iteration 1	2	3	4	5
Small	0.854	0.836	0.866	0.858	0.867
Medium	0.963	0.963	0.943	0.956	0.971
Large	0.955	0.952	0.948	0.946	0.930

Table B.13: Overall Accuracy for Experiment: No Econ – PCA – SVM

Map Size	Iteration 1	2	3	4	5
Small	0.884	0.888	0.901	0.907	0.911
Medium	0.976	0.975	0.973	0.976	0.977
Large	0.978	0.977	0.976	0.976	0.968

Table B.14: Overall Accuracy for Experiment: No Econ – LDA – Knn

Map Size	Iteration 1	2	3	4	5
Small	0.892	0.870	0.916	0.892	0.913
Medium	0.961	0.965	0.956	0.974	0.969
Large	0.972	0.966	0.963	0.952	0.960

Table B.15: Overall Accuracy for Experiment: No Econ – LDA – SVM

Map Size	Iteration 1	2	3	4	5
Small	0.903	0.890	0.926	0.924	0.923
Medium	0.962	0.975	0.967	0.975	0.973
Large	0.972	0.970	0.966	0.964	0.963

B.3 Confusion Matrices

Tables B.16 through B.51 present the 36 confusion matrices from the experiments. The far left column lists the actual strategies, while the top row list the predicted strategies. The second to last column is the total actual samples for each strategy. The second to last row is the total predicted samples for each strategy. The last column gives the producer accuracy (PA). PA is computed as the number of correctly predicted samples for a given strategy divided by the total number of actual samples for the given strategy. The bottom row gives the consumer accuracy (CA). CA is computed as the number of correctly predicted samples for given strategy divided by the total number of predicted samples for the given strategy. The diagonal gives the number of correctly predicted samples for each strategy. The upper and lower triangles give the counts of mis-predicted samples. For example, looking at Table B.16, the cell at (1, 5)—row 1, column 5—is read *of the 14,801 actual IR samples, the classifier mistakenly predicted 32 as Bomber*. This is a producer focus. Alternatively, for a consumer focus, the cell could be read as *of the 12,856 samples the classifier predicted to be Bomber, 32 where actually IR samples*. The number in the bottom corner is the overall accuracy—it is the sum of the diagonal divided by the sum of the entire matrix. Each confusion matrix contains the results of all 5 iterations of 5-fold cross-validation.

Table B.16: Confusion Matrix for Experiment: Econ – None – Knn on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14758	0	0	0	32	11	0	0	14801	99.7%
TR	0	9532	51	196	30	7	0	113	9929	96.0%
Blitz	130	788	9950	325	285	212	0	283	11973	83.1%
Artil	55	863	558	9123	168	35	0	247	11049	82.6%
Bomber	76	74	162	32	9166	652	0	5	10167	90.2%
Antiair	99	49	149	68	3036	7034	0	1	10436	67.4%
Exp	1209	7	29	24	9	0	14179	6	15463	91.7%
Turtle	63	864	1833	256	130	31	0	11795	14972	78.8%
Totals	16390	12177	12732	10024	12856	7982	14179	12450		
CA	90.0%	78.3%	78.1%	91.0%	71.3%	88.1%	100.0%	94.7%		86.6%

Table B.17: Confusion Matrix for Experiment: Econ – None – Knn on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15618	0	0	0	0	0	1	0	15619	100.0%
TR	0	18228	7	1	0	0	0	2	18238	99.9%
Blitz	0	290	17585	208	7	97	0	658	18845	93.3%
Artil	0	147	510	23605	41	77	0	167	24547	96.2%
Bomber	4	69	348	0	10676	283	0	0	11380	93.8%
Antiair	1	13	362	0	1102	11105	0	0	12583	88.3%
Exp	1332	3	15	0	35	19	24406	0	25810	94.6%
Turtle	0	122	1025	108	2	29	0	24644	25930	95.0%
Totals	16955	18872	19852	23922	11863	11610	24407	25471		
CA	92.1%	96.6%	88.6%	98.7%	90.0%	95.7%	100.0%	96.8%		95.4%

Table B.18: Confusion Matrix for Experiment: Econ – None – Knn on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19239	56	0	0	0	0	164	0	19459	98.9%
TR	1	22675	14	6	17	11	0	80	22804	99.4%
Blitz	0	216	22735	81	144	114	0	645	23935	95.0%
Artil	0	340	488	27481	95	87	0	429	28920	95.0%
Bomber	0	35	426	18	12903	474	0	7	13863	93.1%
Antiair	0	46	427	18	1201	14556	0	11	16259	89.5%
Exp	521	65	22	4	45	7	33606	14	34284	98.0%
Turtle	0	94	487	807	54	68	0	29062	30572	95.1%
Totals	19761	23527	24599	28415	14459	15317	33770	30248		
CA	97.4%	96.4%	92.4%	96.7%	89.2%	95.0%	99.5%	96.1%		95.9%

Table B.19: Confusion Matrix for Experiment: Econ – None – SVM on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14627	0	0	9	57	0	108	0	14801	98.8%
TR	0	9530	21	210	145	0	0	23	9929	96.0%
Blitz	0	333	10401	66	855	1	0	317	11973	86.9%
Artil	0	505	502	8999	535	16	0	492	11049	81.4%
Bomber	0	12	71	158	9896	30	0	0	10167	97.3%
Antiair	0	3	62	154	2193	8024	0	0	10436	76.9%
Exp	169	0	4	46	2	39	15203	0	15463	98.3%
Turtle	0	399	1357	383	209	48	0	12576	14972	84.0%
Totals	14796	10782	12418	10025	13892	8158	15311	13408		
CA	98.9%	88.4%	83.8%	89.8%	71.2%	98.4%	99.3%	93.8%		90.3%

Table B.20: Confusion Matrix for Experiment: Econ – None – SVM on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15619	0	0	0	0	0	0	0	15619	100.0%
TR	1	18217	0	0	4	0	0	16	18238	99.9%
Blitz	0	13	18335	7	446	0	0	44	18845	97.3%
Artil	0	8	24	24127	44	37	0	307	24547	98.3%
Bomber	0	26	0	0	11354	0	0	0	11380	99.8%
Antiair	0	2	0	0	1025	11556	0	0	12583	91.8%
Exp	105	0	0	1	14	24	25666	0	25810	99.4%
Turtle	0	0	95	72	39	2	0	25722	25930	99.2%
Totals	15725	18266	18454	24207	12926	11619	25666	26089		
CA	99.3%	99.7%	99.4%	99.7%	87.8%	99.5%	100.0%	98.6%		98.5%

Table B.21: Confusion Matrix for Experiment: Econ – None – SVM on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19434	24	0	0	0	0	1	0	19459	99.9%
TR	14	22601	0	0	51	0	0	138	22804	99.1%
Blitz	0	16	23188	9	686	0	0	36	23935	96.9%
Artil	7	18	86	28083	343	23	0	360	28920	97.1%
Bomber	21	5	81	0	13756	0	0	0	13863	99.2%
Antiair	32	0	84	0	1269	14874	0	0	16259	91.5%
Exp	31	26	2	23	19	16	34167	0	34284	99.7%
Turtle	0	7	3	313	302	1	0	29946	30572	98.0%
Totals	19539	22697	23444	28428	16426	14914	34168	30480		
CA	99.5%	99.6%	98.9%	98.8%	83.7%	99.7%	100.0%	98.2%		97.9%

Table B.22: Confusion Matrix for Experiment: Econ – PCA – Knn on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14800	0	0	0	1	0	0	0	14801	100.0%
TR	27	9289	13	594	1	5	0	0	9929	93.6%
Blitz	112	849	9631	540	259	190	0	392	11973	80.4%
Artil	53	737	479	9343	178	72	0	187	11049	84.6%
Bomber	202	74	117	31	9043	699	0	1	10167	88.9%
Antiair	245	78	112	117	2897	6982	0	5	10436	66.9%
Exp	1393	2	11	42	6	9	13995	5	15463	90.5%
Turtle	60	659	2320	717	90	56	0	11070	14972	73.9%
Totals	16892	11688	12683	11384	12475	8013	13995	11660		
CA	87.6%	79.5%	75.9%	82.1%	72.5%	87.1%	100.0%	94.9%		85.2%

Table B.23: Confusion Matrix for Experiment: Econ – PCA – Knn on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15560	53	0	0	1	2	3	0	15619	99.6%
TR	13	18179	3	2	40	0	0	1	18238	99.7%
Blitz	0	387	17110	398	77	106	0	767	18845	90.8%
Artil	0	135	451	23727	34	95	0	105	24547	96.7%
Bomber	7	65	276	0	10456	576	0	0	11380	91.9%
Antiair	1	13	269	0	1846	10454	0	0	12583	83.1%
Exp	1361	2	10	0	20	48	24369	0	25810	94.4%
Turtle	0	180	1044	254	8	42	0	24402	25930	94.1%
Totals	16942	19014	19163	24381	12482	11323	24372	25275		
CA	91.8%	95.6%	89.3%	97.3%	83.8%	92.3%	100.0%	96.5%		94.3%

Table B.24: Confusion Matrix for Experiment: Econ – PCA – Knn on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19277	28	0	0	0	0	154	0	19459	99.1%
TR	7	22631	1	0	38	11	0	116	22804	99.2%
Blitz	0	453	22566	62	207	195	0	452	23935	94.3%
Artil	7	316	820	27007	123	120	0	527	28920	93.4%
Bomber	0	42	324	1	12545	951	0	0	13863	90.5%
Antiair	43	38	331	21	2686	13140	0	0	16259	80.8%
Exp	979	48	8	13	14	51	33171	0	34284	96.8%
Turtle	0	169	1267	679	83	63	0	28311	30572	92.6%
Totals	20313	23725	25317	27783	15696	14531	33325	29406		
CA	94.9%	95.4%	89.1%	97.2%	79.9%	90.4%	99.5%	96.3%		94.0%

Table B.25: Confusion Matrix for Experiment: Econ – PCA – SVM on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14673	0	0	9	57	0	62	0	14801	99.1%
TR	0	9198	26	260	145	3	0	297	9929	92.6%
Blitz	0	212	10043	132	893	80	0	613	11973	83.9%
Artil	0	345	266	9011	440	233	0	754	11049	81.6%
Bomber	0	0	55	127	9200	785	0	0	10167	90.5%
Antiair	0	0	41	150	1922	8321	0	2	10436	79.7%
Exp	233	0	1	25	2	54	15148	0	15463	98.0%
Turtle	0	294	232	130	160	130	0	14026	14972	93.7%
Totals	14906	10049	10664	9844	12819	9606	15210	15692		
CA	98.4%	91.5%	94.2%	91.5%	71.8%	86.6%	99.6%	89.4%		90.7%

Table B.26: Confusion Matrix for Experiment: Econ – PCA – SVM on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15617	0	0	0	2	0	0	0	15619	100.0%
TR	0	18220	0	0	1	1	0	16	18238	99.9%
Blitz	0	20	18329	0	443	0	0	53	18845	97.3%
Artil	0	5	23	24160	52	26	0	281	24547	98.4%
Bomber	0	52	0	0	11320	8	0	0	11380	99.5%
Antiair	0	12	0	1	1160	11410	0	0	12583	90.7%
Exp	63	0	0	0	15	7	25725	0	25810	99.7%
Turtle	0	0	15	160	39	0	0	25716	25930	99.2%
Totals	15680	18309	18367	24321	13032	11452	25725	26066		
CA	99.6%	99.5%	99.8%	99.3%	86.9%	99.6%	100.0%	98.7%		98.4%

Table B.27: Confusion Matrix for Experiment: Econ – PCA – SVM on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19310	23	0	0	0	0	126	0	19459	99.2%
TR	10	22595	0	0	0	51	0	148	22804	99.1%
Blitz	0	16	23129	4	742	16	0	28	23935	96.6%
Artil	7	14	69	28109	321	44	0	356	28920	97.2%
Bomber	12	1	0	1	13754	95	0	0	13863	99.2%
Antiair	43	3	0	6	1976	14231	0	0	16259	87.5%
Exp	23	18	1	5	1	56	34180	0	34284	99.7%
Turtle	0	0	4	230	277	27	3	30031	30572	98.2%
Totals	19405	22670	23203	28355	17071	14520	34309	30563		
CA	99.5%	99.7%	99.7%	99.1%	80.6%	98.0%	99.6%	98.3%		97.5%

Table B.28: Confusion Matrix for Experiment: Econ – LDA – Knn on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14669	1	0	11	27	0	93	0	14801	99.1%
TR	3	9569	111	147	28	2	0	69	9929	96.4%
Blitz	57	415	10344	247	255	83	0	572	11973	86.4%
Artil	33	596	442	9393	96	58	1	430	11049	85.0%
Bomber	70	123	160	35	8789	909	1	80	10167	86.4%
Antiair	94	65	157	136	1744	8126	1	113	10436	77.9%
Exp	359	4	7	15	6	7	15044	21	15463	97.3%
Turtle	58	573	647	466	70	58	0	13100	14972	87.5%
Totals	15343	11346	11868	10450	11015	9243	15140	14385		
CA	95.6%	84.3%	87.2%	89.9%	79.8%	87.9%	99.4%	91.1%		90.1%

Table B.29: Confusion Matrix for Experiment: Econ – LDA – Knn on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15568	17	0	1	0	0	29	4	15619	99.7%
TR	20	18183	0	1	0	0	12	22	18238	99.7%
Blitz	0	133	17784	22	1	110	5	790	18845	94.4%
Artil	5	175	176	23856	4	8	44	279	24547	97.2%
Bomber	0	0	3	11	10811	542	7	6	11380	95.0%
Antiair	11	2	5	22	688	11827	27	1	12583	94.0%
Exp	198	9	2	6	1	4	25573	17	25810	99.1%
Turtle	2	535	745	264	0	12	4	24368	25930	94.0%
Totals	15804	19054	18715	24183	11505	12503	25701	25487		
CA	98.5%	95.4%	95.0%	98.6%	94.0%	94.6%	99.5%	95.6%		96.7%

Table B.30: Confusion Matrix for Experiment: Econ – LDA – Knn on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19430	0	0	26	0	0	2	1	19459	99.9%
TR	9	22663	10	12	3	0	2	105	22804	99.4%
Blitz	0	166	22344	138	209	466	5	607	23935	93.4%
Artil	19	326	245	27700	78	79	27	446	28920	95.8%
Bomber	26	9	8	83	13359	346	1	31	13863	96.4%
Antiair	6	8	12	79	1065	15028	10	51	16259	92.4%
Exp	175	27	9	13	17	0	34037	6	34284	99.3%
Turtle	0	618	1206	777	8	59	10	27894	30572	91.2%
Totals	19665	23817	23834	28828	14739	15978	34094	29141		
CA	98.8%	95.2%	93.7%	96.1%	90.6%	94.1%	99.8%	95.7%		96.0%

Table B.31: Confusion Matrix for Experiment: Econ – LDA – SVM on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14641	0	0	4	32	1	123	0	14801	98.9%
TR	0	9524	65	287	25	0	13	15	9929	95.9%
Blitz	10	137	10683	482	146	138	135	242	11973	89.2%
Artil	1	235	287	9717	106	83	230	390	11049	87.9%
Bomber	20	28	150	150	8853	896	68	2	10167	87.1%
Antiair	10	12	69	121	1164	8757	284	19	10436	83.9%
Exp	131	2	7	46	1	3	15270	3	15463	98.8%
Turtle	13	115	228	727	33	47	879	12930	14972	86.4%
Totals	14826	10053	11489	11534	10360	9925	17002	13601		
CA	98.8%	94.7%	93.0%	84.2%	85.5%	88.2%	89.8%	95.1%		91.5%

Table B.32: Confusion Matrix for Experiment: Econ – LDA – SVM on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15338	0	0	202	0	0	33	46	15619	98.2%
TR	0	18194	0	11	0	0	0	33	18238	99.8%
Blitz	0	0	18238	341	1	0	0	265	18845	96.8%
Artil	0	5	32	24337	1	8	8	156	24547	99.1%
Bomber	0	0	0	519	10281	414	0	166	11380	90.3%
Antiair	0	0	0	185	381	11403	2	612	12583	90.6%
Exp	27	0	0	478	0	8	25158	139	25810	97.5%
Turtle	0	0	44	483	0	0	0	25403	25930	98.0%
Totals	15365	18199	18314	26556	10664	11833	25201	26820		
CA	99.8%	100.0%	99.6%	91.6%	96.4%	96.4%	99.8%	94.7%		97.0%

Table B.33: Confusion Matrix for Experiment: Econ – LDA – SVM on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19207	0	0	17	0	0	32	203	19459	98.7%
TR	4	22719	0	18	0	0	26	37	22804	99.6%
Blitz	0	0	23187	35	27	70	0	616	23935	96.9%
Artil	0	6	34	28186	2	8	11	673	28920	97.5%
Bomber	4	0	97	7	12844	382	14	515	13863	92.6%
Antiair	0	0	102	12	599	14382	1	1163	16259	88.5%
Exp	13	0	0	44	3	1	33600	623	34284	98.0%
Turtle	0	0	10	565	2	0	0	29995	30572	98.1%
Totals	19228	22725	23430	28884	13477	14843	33684	33825		
CA	99.9%	100.0%	99.0%	97.6%	95.3%	96.9%	99.8%	88.7%		96.9%

Table B.34: Confusion Matrix for Experiment: No Econ – None – Knn on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14759	0	0	34	0	0	8	0	14801	99.7%
TR	2	9442	25	460	0	0	0	0	9929	95.1%
Blitz	129	802	9832	443	416	121	0	230	11973	82.1%
Artil	51	935	422	9313	121	12	0	195	11049	84.3%
Bomber	171	80	278	72	9167	399	0	0	10167	90.2%
Antiair	126	27	266	213	2368	7436	0	0	10436	71.3%
Exp	1193	15	0	52	4	0	14195	4	15463	91.8%
Turtle	64	643	1879	585	118	21	0	11662	14972	77.9%
Totals	16495	11944	12702	11172	12194	7989	14203	12091		
CA	89.5%	79.1%	77.4%	83.4%	75.2%	93.1%	99.9%	96.5%		86.9%

Table B.35: Confusion Matrix for Experiment: No Econ – None – Knn on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15590	29	0	0	0	0	0	0	15619	99.8%
TR	0	18238	0	0	0	0	0	0	18238	100.0%
Blitz	0	543	18043	0	89	0	0	170	18845	95.7%
Artil	0	352	289	23704	12	16	0	174	24547	96.6%
Bomber	2	4	352	0	11022	0	0	0	11380	96.9%
Antiair	6	24	366	0	852	11335	0	0	12583	90.1%
Exp	993	56	12	0	0	6	24743	0	25810	95.9%
Turtle	0	482	795	129	0	29	0	24495	25930	94.5%
Totals	16591	19728	19857	23833	11975	11386	24743	24839		
CA	94.0%	92.4%	90.9%	99.5%	92.0%	99.6%	100.0%	98.6%		96.2%

Table B.36: Confusion Matrix for Experiment: No Econ – None – Knn on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19427	32	0	0	0	0	0	0	19459	99.8%
TR	12	22792	0	0	0	0	0	0	22804	99.9%
Blitz	0	747	22868	9	307	0	0	4	23935	95.5%
Artil	27	625	505	27171	130	5	0	457	28920	94.0%
Bomber	32	12	464	0	13251	104	0	0	13863	95.6%
Antiair	16	7	456	9	1088	14672	0	11	16259	90.2%
Exp	1173	87	18	15	0	0	32959	32	34284	96.1%
Turtle	0	600	690	719	124	3	0	28436	30572	93.0%
Totals	20687	24902	25001	27923	14900	14784	32959	28940		
CA	93.9%	91.5%	91.5%	97.3%	88.9%	99.2%	100.0%	98.3%		95.5%

Table B.37: Confusion Matrix for Experiment: No Econ – None – SVM on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14607	0	0	9	57	0	128	0	14801	98.7%
TR	0	9523	11	163	203	0	0	29	9929	95.9%
Blitz	0	361	10272	27	1057	0	0	256	11973	85.8%
Artil	0	537	476	8859	669	26	0	482	11049	80.2%
Bomber	0	0	11	165	9991	0	0	0	10167	98.3%
Antiair	0	0	6	140	2267	8023	0	0	10436	76.9%
Exp	161	0	0	31	6	39	15226	0	15463	98.5%
Turtle	0	430	1364	40	241	54	0	12843	14972	85.8%
Totals	14768	10851	12140	9434	14491	8142	15354	13610		
CA	98.9%	87.8%	84.6%	93.9%	68.9%	98.5%	99.2%	94.4%		90.4%

Table B.38: Confusion Matrix for Experiment: No Econ – None – SVM on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15619	0	0	0	0	0	0	0	15619	100.0%
TR	26	18075	0	0	137	0	0	0	18238	99.1%
Blitz	0	12	18370	6	446	0	0	11	18845	97.5%
Artil	2	233	29	24078	53	57	0	95	24547	98.1%
Bomber	0	0	0	0	11380	0	0	0	11380	100.0%
Antiair	7	1	0	0	1030	11545	0	0	12583	91.8%
Exp	131	1	0	0	8	19	25651	0	25810	99.4%
Turtle	0	314	718	150	39	7	0	24702	25930	95.3%
Totals	15785	18636	19117	24234	13093	11628	25651	24808		
CA	98.9%	97.0%	96.1%	99.4%	86.9%	99.3%	100.0%	99.6%		97.7%

Table B.39: Confusion Matrix for Experiment: No Econ – None – SVM on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19416	0	0	0	43	0	0	0	19459	99.8%
TR	13	22660	2	0	129	0	0	0	22804	99.4%
Blitz	0	4	23147	11	758	0	0	15	23935	96.7%
Artil	16	255	77	28008	378	9	0	177	28920	96.8%
Bomber	30	0	0	0	13833	0	0	0	13863	99.8%
Antiair	19	0	0	0	1340	14900	0	0	16259	91.6%
Exp	34	0	0	20	42	15	34173	0	34284	99.7%
Turtle	0	323	0	243	304	1	0	29701	30572	97.2%
Totals	19528	23242	23226	28282	16827	14925	34173	29893		
CA	99.4%	97.5%	99.7%	99.0%	82.2%	99.8%	100.0%	99.4%		97.8%

Table B.40: Confusion Matrix for Experiment: No Econ – PCA – Knn on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14768	0	0	0	33	0	0	0	14801	99.8%
TR	1	9535	96	256	26	15	0	0	9929	96.0%
Blitz	95	1122	8738	491	662	59	0	806	11973	73.0%
Artil	30	958	379	9220	239	22	0	201	11049	83.4%
Bomber	133	41	91	55	9672	175	0	0	10167	95.1%
Antiair	126	13	103	65	2957	7168	0	4	10436	68.7%
Exp	1482	0	1	12	18	0	13903	47	15463	89.9%
Turtle	49	764	1931	581	109	27	0	11511	14972	76.9%
Totals	16684	12433	11339	10680	13716	7466	13903	12569		
CA	88.5%	76.7%	77.1%	86.3%	70.5%	96.0%	100.0%	91.6%		85.6%

Table B.41: Confusion Matrix for Experiment: No Econ – PCA – Knn on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15619	0	0	0	0	0	0	0	15619	100.0%
TR	0	18177	39	0	7	15	0	0	18238	99.7%
Blitz	0	689	17719	0	176	0	0	261	18845	94.0%
Artil	0	568	131	23616	41	52	0	139	24547	96.2%
Bomber	6	9	263	0	11102	0	0	0	11380	97.6%
Antiair	9	6	272	0	970	11326	0	0	12583	90.0%
Exp	1119	9	5	0	23	6	24648	0	25810	95.5%
Turtle	0	561	774	54	16	67	0	24458	25930	94.3%
Totals	16753	20019	19203	23670	12335	11466	24648	24858		
CA	93.2%	90.8%	92.3%	99.8%	90.0%	98.8%	100.0%	98.4%		95.9%

Table B.42: Confusion Matrix for Experiment: No Econ – PCA – Knn on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19427	32	0	0	0	0	0	0	19459	99.8%
TR	14	22721	17	0	33	19	0	0	22804	99.6%
Blitz	0	707	22735	10	303	180	0	0	23935	95.0%
Artil	23	650	481	27106	186	137	0	337	28920	93.7%
Bomber	39	14	268	0	13314	228	0	0	13863	96.0%
Antiair	47	5	262	39	1222	14682	0	2	16259	90.3%
Exp	1767	41	5	33	33	22	32383	0	34284	94.5%
Turtle	0	658	1417	663	177	130	0	27527	30572	90.0%
Totals	21317	24828	25185	27851	15268	15398	32383	27866		
CA	91.1%	91.5%	90.3%	97.3%	87.2%	95.4%	100.0%	98.8%		94.6%

Table B.43: Confusion Matrix for Experiment: No Econ – PCA – SVM on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14628	0	0	0	57	9	107	0	14801	98.8%
TR	0	9132	345	0	364	1	0	87	9929	92.0%
Blitz	0	383	10178	1	1054	10	0	347	11973	85.0%
Artil	0	374	759	8821	653	47	0	395	11049	79.8%
Bomber	0	0	61	0	9876	230	0	0	10167	97.1%
Antiair	0	0	47	92	2250	8045	0	2	10436	77.1%
Exp	228	0	5	1	6	70	15151	2	15463	98.0%
Turtle	0	294	1459	22	187	134	0	12876	14972	86.0%
Totals	14856	10183	12854	8937	14447	8546	15258	13709		
CA	98.5%	89.7%	79.2%	98.7%	68.4%	94.1%	99.3%	93.9%		89.8%

Table B.44: Confusion Matrix for Experiment: No Econ – PCA – SVM on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15619	0	0	0	0	0	0	0	15619	100.0%
TR	22	18066	0	0	32	118	0	0	18238	99.1%
Blitz	0	419	17980	0	446	0	0	0	18845	95.4%
Artil	0	222	30	24112	54	41	0	88	24547	98.2%
Bomber	6	0	0	0	11374	0	0	0	11380	99.9%
Antiair	6	0	0	0	1013	11564	0	0	12583	91.9%
Exp	76	0	0	0	0	9	25725	0	25810	99.7%
Turtle	0	311	711	107	33	8	0	24760	25930	95.5%
Totals	15729	19018	18721	24219	12952	11740	25725	24848		
CA	99.3%	95.0%	96.0%	99.6%	87.8%	98.5%	100.0%	99.6%		97.5%

Table B.45: Confusion Matrix for Experiment: No Econ – PCA – SVM on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19416	0	0	0	2	41	0	0	19459	99.8%
TR	17	22660	2	0	60	65	0	0	22804	99.4%
Blitz	0	74	22723	18	757	3	0	360	23935	94.9%
Artil	12	255	36	27993	378	19	0	227	28920	96.8%
Bomber	31	0	0	0	13818	14	0	0	13863	99.7%
Antiair	32	0	0	0	1460	14767	0	0	16259	90.8%
Exp	29	0	0	0	4	72	34179	0	34284	99.7%
Turtle	0	319	0	212	304	1	0	29736	30572	97.3%
Totals	19537	23308	22761	28223	16783	14982	34179	30323		
CA	99.4%	97.2%	99.8%	99.2%	82.3%	98.6%	100.0%	98.1%		97.5%

Table B.46: Confusion Matrix for Experiment: No Econ – LDA – Knn on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14673	0	11	0	20	0	97	0	14801	99.1%
TR	0	9699	70	101	27	0	0	32	9929	97.7%
Blitz	59	551	10051	282	552	2	0	476	11973	83.9%
Artil	28	768	264	9420	127	0	0	442	11049	85.3%
Bomber	104	87	331	37	8883	643	0	82	10167	87.4%
Antiair	105	62	374	88	1917	7832	0	58	10436	75.0%
Exp	395	9	0	5	0	0	15015	39	15463	97.1%
Turtle	58	831	461	553	60	60	0	12949	14972	86.5%
Totals	15422	12007	11562	10486	11586	8537	15112	14078		
CA	95.1%	80.8%	86.9%	89.8%	76.7%	91.7%	99.4%	92.0%		89.6%

Table B.47: Confusion Matrix for Experiment: No Econ – LDA – Knn on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15594	0	0	0	0	0	25	0	15619	99.8%
TR	1	18179	0	0	0	0	22	36	18238	99.7%
Blitz	0	115	17891	153	353	1	0	332	18845	94.9%
Artil	3	232	262	23692	9	49	4	296	24547	96.5%
Bomber	0	3	90	0	10714	572	0	1	11380	94.1%
Antiair	10	16	91	5	706	11740	4	11	12583	93.3%
Exp	370	7	0	5	0	8	25415	5	25810	98.5%
Turtle	2	699	607	181	0	44	0	24397	25930	94.1%
Totals	15980	19251	18941	24036	11782	12414	25470	25078		
CA	97.6%	94.4%	94.5%	98.6%	90.9%	94.6%	99.8%	97.3%		96.5%

Table B.48: Confusion Matrix for Experiment: No Econ – LDA – Knn on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19430	26	0	0	0	0	3	0	19459	99.9%
TR	10	22773	0	21	0	0	0	0	22804	99.9%
Blitz	1	139	22801	14	273	1	0	706	23935	95.3%
Artil	40	269	210	27468	22	11	3	897	28920	95.0%
Bomber	27	20	410	0	13108	202	0	96	13863	94.6%
Antiair	7	7	421	1	890	14826	12	95	16259	91.2%
Exp	179	27	5	6	0	0	34042	25	34284	99.3%
Turtle	6	555	968	450	0	2	4	28587	30572	93.5%
Totals	19700	23816	24815	27960	14293	15042	34064	30406		
CA	98.6%	95.6%	91.9%	98.2%	91.7%	98.6%	99.9%	94.0%		96.3%

Table B.49: Confusion Matrix for Experiment: No Econ – LDA – SVM on Small Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	14619	0	0	12	45	0	125	0	14801	98.8%
TR	0	9493	32	354	26	0	15	9	9929	95.6%
Blitz	6	190	10122	491	405	340	94	325	11973	84.5%
Artil	0	374	278	9590	128	51	64	564	11049	86.8%
Bomber	28	36	86	223	9001	728	64	1	10167	88.5%
Antiair	11	9	66	197	1373	8491	247	42	10436	81.4%
Exp	169	3	1	47	0	2	15237	4	15463	98.5%
Turtle	0	206	183	662	56	31	222	13612	14972	90.9%
Totals	14833	10311	10768	11576	11034	9643	16068	14557		
CA	98.6%	92.1%	94.0%	82.8%	81.6%	88.1%	94.8%	93.5%		91.3%

Table B.50: Confusion Matrix for Experiment: No Econ – LDA – SVM on Medium Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	15535	0	0	82	0	0	2	0	15619	99.5%
TR	0	18195	0	15	0	20	0	8	18238	99.8%
Blitz	0	4	17996	421	171	184	0	69	18845	95.5%
Artil	0	16	24	24198	2	14	7	286	24547	98.6%
Bomber	0	1	91	395	10709	184	0	0	11380	94.1%
Antiair	0	14	91	547	738	11191	1	1	12583	88.9%
Exp	90	7	0	394	0	14	25305	0	25810	98.0%
Turtle	0	0	50	555	0	0	0	25325	25930	97.7%
Totals	15625	18237	18252	26607	11620	11607	25315	25689		
CA	99.4%	99.8%	98.6%	90.9%	92.2%	96.4%	100.0%	98.6%		97.1%

Table B.51: Confusion Matrix for Experiment: No Econ – LDA – SVM on Large Map

	Predicted									
	IR	TR	Blitz	Artil	Bomber	Antiair	Exp	Turtle	Samples	PA
IR	19336	0	0	49	18	23	5	28	19459	99.4%
TR	5	22766	0	30	0	0	3	0	22804	99.8%
Blitz	0	0	22966	556	275	0	0	138	23935	96.0%
Artil	7	17	16	28464	1	0	7	408	28920	98.4%
Bomber	14	0	412	207	12963	9	0	258	13863	93.5%
Antiair	5	1	401	540	833	14009	0	470	16259	86.2%
Exp	30	7	0	328	0	3	33629	287	34284	98.1%
Turtle	0	0	5	882	0	0	0	29685	30572	97.1%
Totals	19397	22791	23800	31056	14090	14044	33644	31274		
CA	99.7%	99.9%	96.5%	91.7%	92.0%	99.8%	100.0%	94.9%		96.7%

B.4 Accuracy Over Time

Figures B.1 through B.12 plot the accuracy over time of the experiments for each of the 12 combinations of data set, feature reduction technique, and classifier.

Figure B.1: Accuracy over Time for Experiment: Econ – None – Knn

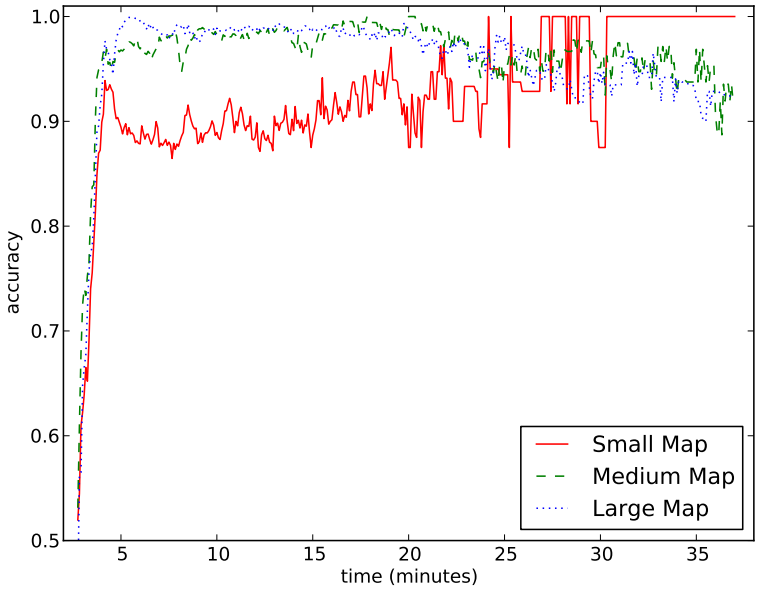


Figure B.2: Accuracy over Time for Experiment: Econ – None – SVM

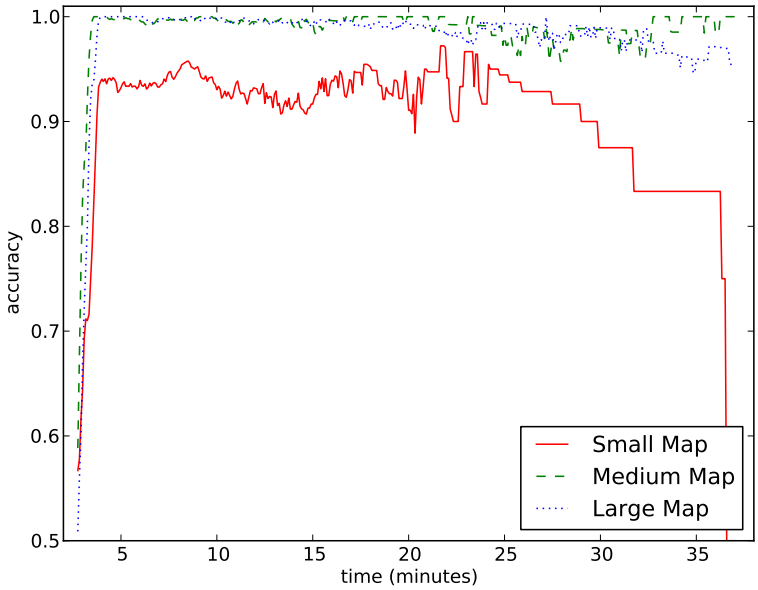


Figure B.3: Accuracy over Time for Experiment: Econ – PCA – Knn

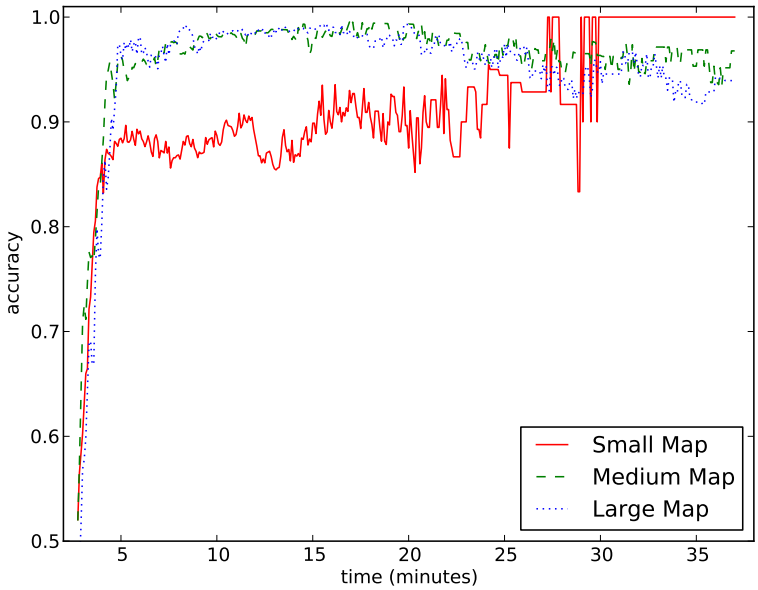


Figure B.4: Accuracy over Time for Experiment: Econ – PCA – SVM

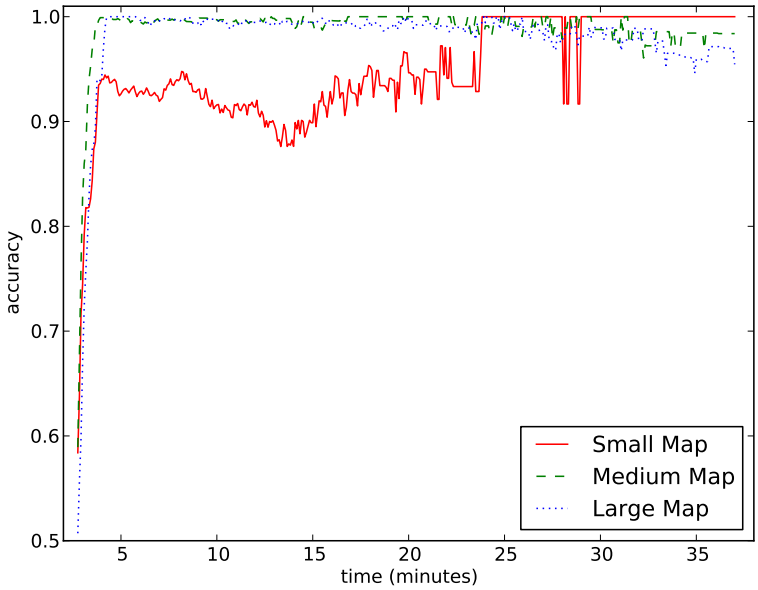


Figure B.5: Accuracy over Time for Experiment: Econ – LDA – Knn

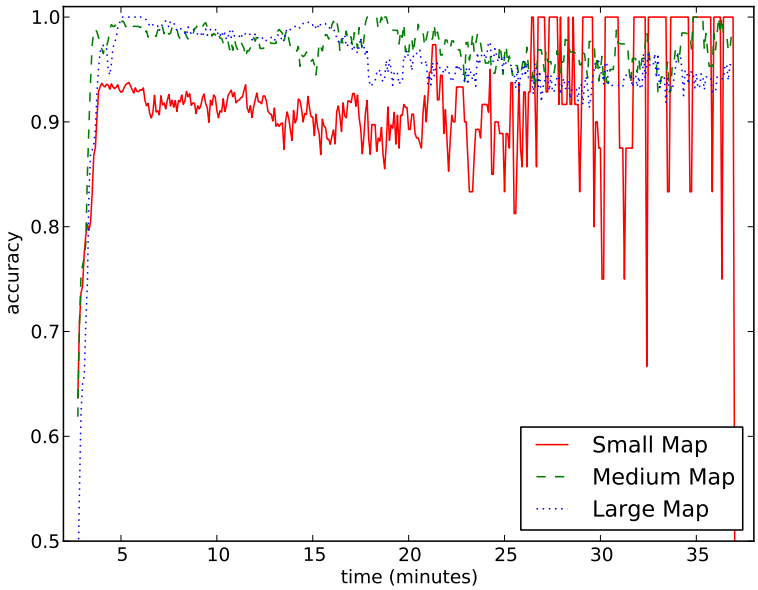


Figure B.6: Accuracy over Time for Experiment: Econ – LDA – SVM

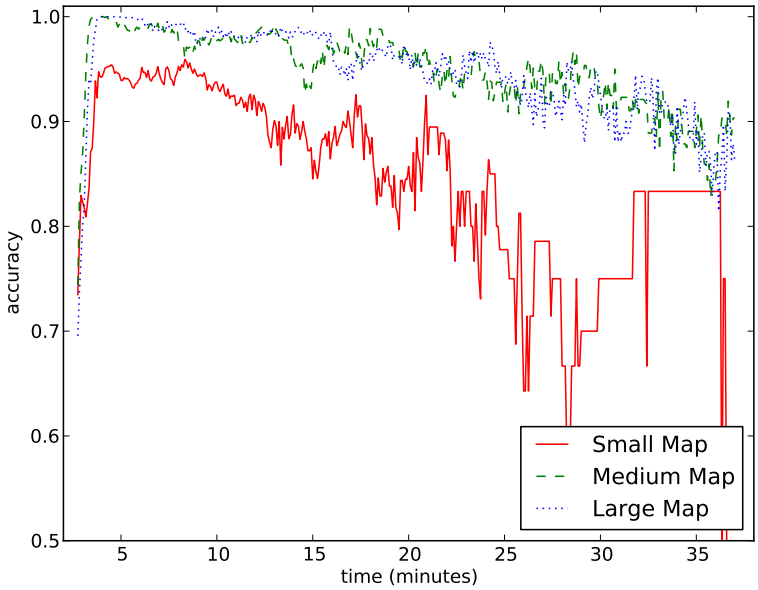


Figure B.7: Accuracy over Time for Experiment: No Econ – None – Knn

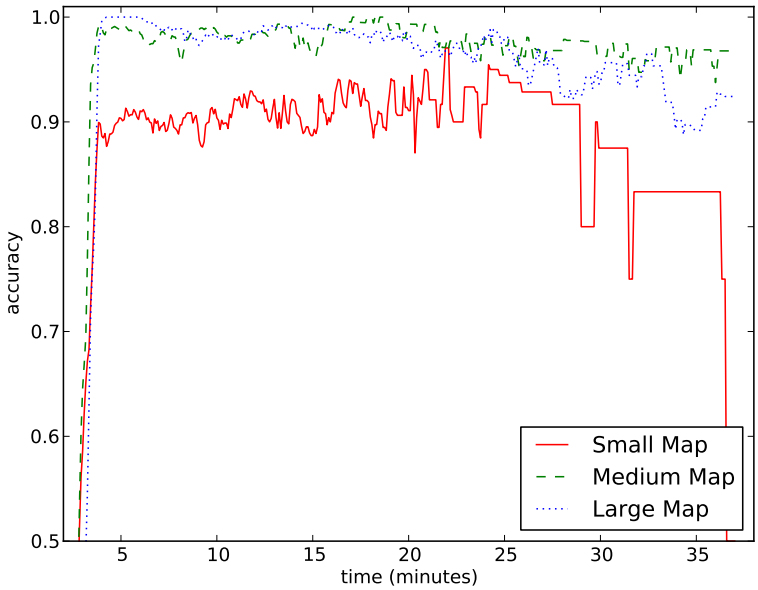


Figure B.8: Accuracy over Time for Experiment: No Econ – None – SVM

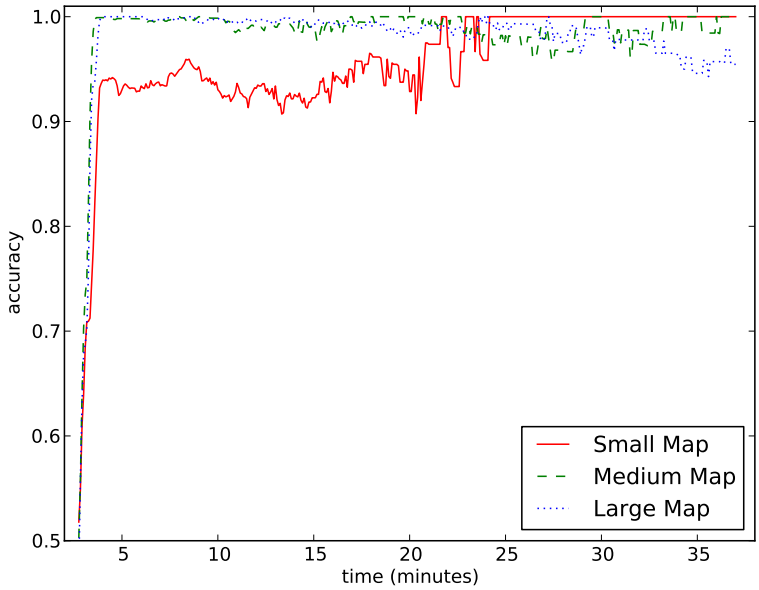


Figure B.9: Accuracy over Time for Experiment: No Econ – PCA – Knn

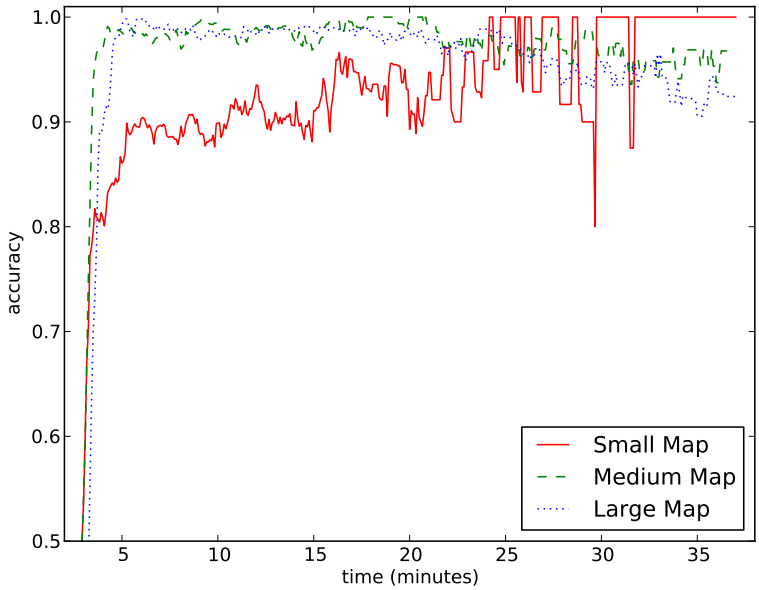


Figure B.10: Accuracy over Time for Experiment: No Econ – PCA – SVM

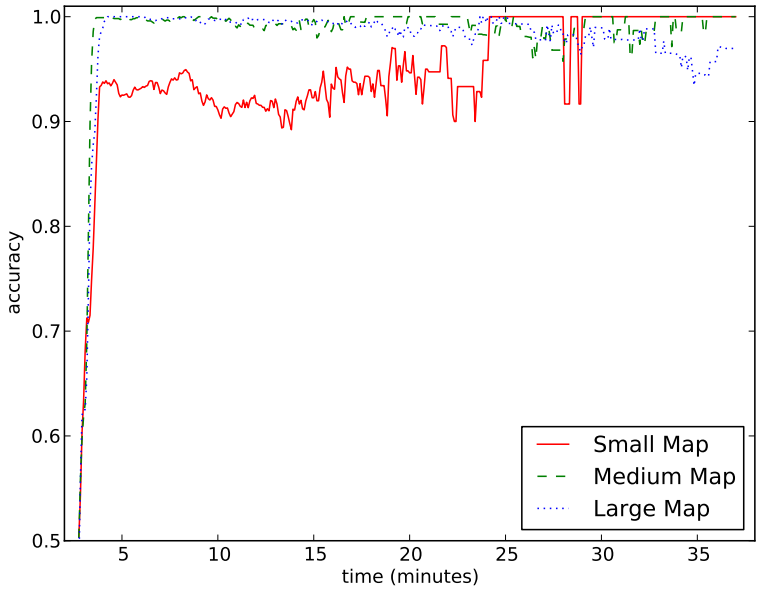


Figure B.11: Accuracy over Time for Experiment: No Econ – LDA – Knn

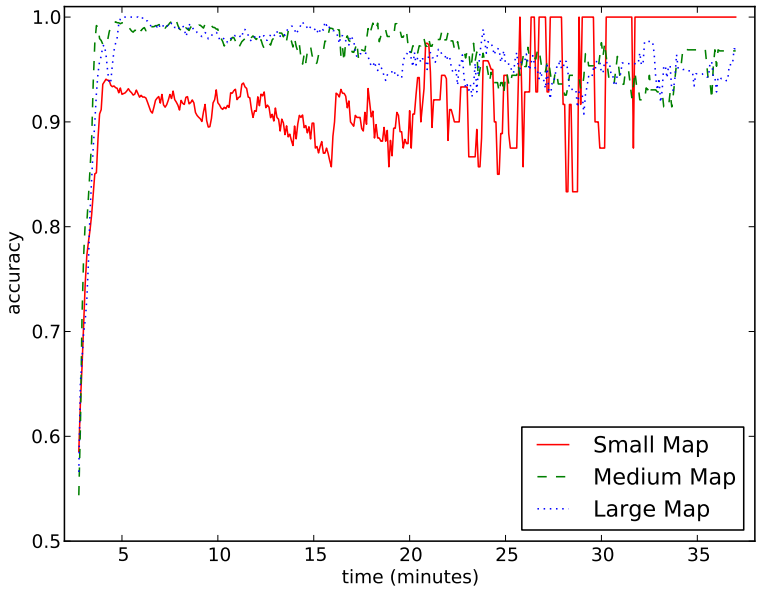
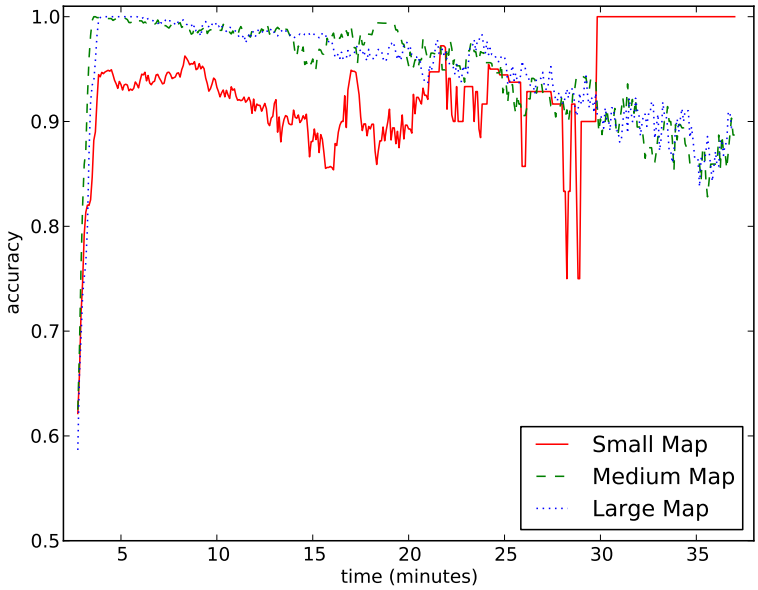


Figure B.12: Accuracy over Time for Experiment: No Econ – LDA – SVM



Bibliography

[1] Aha, David W., Matthew Molineaux, and Marc Ponsen. "Learning to win: Case-based plan selection in a real-time strategy game". *in Proceedings of the Sixth International Conference on Case-Based Reasoning*, 5–20. Springer, 2005.

[2] Association for the Advancement of Artificial Intelligence (AAAI). "AIIDE 2011: AI and Interactive Digital Entertainment Conference Accepted Papers", 2011. URL http://www.movingai.com/aiide11/papers.html. [Online; accessed 24-Oct-2011].

[3] Bakkes, S and P Spronck. "Phase-dependent evaluation in RTS games". *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence*, 2007.

[4] Bakkes, S, P Spronck, and J Van Den Herik. "Rapid and Reliable Adaptation of Video Game AI". *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):93–104, 2009.

[5] Bakkes, Sander C. J., Pieter H. M. Spronck, and H. Jaap van den Herik. "Opponent modelling for case-based adaptive game AI". *Entertainment Computing*, 1(1):27–37, January 2009. ISSN 18759521.

[6] Balla, Radha-Krishna and Alan Fern. "UCT for tactical assault planning in real-time strategy games". *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, 40–45. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.

[7] Barr, Richard S., Bruce L. Golden, James P. Kelly, Mauricio G. C, Resende William, and R. Stewart. "Designing and reporting on computational experiments with heuristic methods". *Journal of Heuristics*, 1:9–32, 1995.

[8] Buro, Michael. "Call for AI research in RTS games". *In Proceedings of the AAAI Workshop on AI in Games*, 139–141. AAAI Press, 2004.

[9] Buro, Michael. "ORTS - A Free Software RTS Game Engine", 2011. URL http://skatgame.net/mburo/orts. [Online; accessed 24-Oct-2011].

[10] Buro, Michael; University of Alberta. "The 2nd Annual AIIDE Starcraft AI Competition", 2011. URL http://skat.dnsalias.net/mburo/sc2011/. [Online; accessed 24-Oct-2011].

[11] Caffrey, Jr., Matthew. "Toward a History-Based Doctrine for Wargaming". *Aerospace Power Journal*, XIV:33–56, 2000.

[12] Chung, Michael, Michael Buro, and Jonathan Schaeffer. "Monte Carlo Planning in RTS Games". *In CIG 2005 Colchester UK*, 117–124, 2005.

[13] Cortes, Corinna and Vladimir Vapnik. “Support-vector networks”. *Machine Learning*, 20(3):273–297, 1995. URL http://www.springerlink.com/index/10.1007/BF00994018.

[14] Duda, R.O., P.E. Hart, and D.G. Stork. *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley, second edition, 2000. ISBN 9780471056690.

[15] Enthought, Inc. “SciPy”, 2011. URL http://www.scipy.org. [Online; accessed 24-Oct-2011].

[16] Gomaa, Hassan. *Designing Concurrent, Distributed, and Real-time Applications with UML*. Addison Wesley, 2000.

[17] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, second edition, 2009.

[18] Heinermann, Adam. “bwapi — An API for interacting with Starcraft: Broodwar (1.16.1) — Google Project Hosting”, 2011. URL https://code.google.com/p/bwapi. [Online; accessed 24-Oct-2011].

[19] IT University Copenhagen, Denmark. “CIG 2010 Accepted Papers”, 2010. URL http://game.itu.dk/cig2010/?page_id=715. [Online; accessed 24-Oct-2011].

[20] Laird, John E. “Using a Computer Game to Develop Advanced AI”. *Computer*, 34(7):70–75, July 2001. ISSN 0018-9162.

[21] Laird, John E. and Michael van Lent. “Human-Level AI’s Killer Application: Interactive Computer Games”. *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 1171–1178. AAAI Press, 2000. ISBN 0-262-51112-6.

[22] Laviers, Kennard and Gita Sukthankar. “A Real-Time Opponent Modeling System for Rush Football”. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2476–2481. 2011.

[23] Laviers, Kennard, Gita Sukthankar, David W. Aha, and Matthew Molineaux. “Improving Offensive Performance Through Opponent Modeling”. *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2009.

[24] Mahlmann, Tobias; IT University Copenhagen. “CIG 2011 StarCraft RTS AI Competition”, 2011. URL http://ls11-www.cs.uni-dortmund.de/rts-competition/starcraft%-cig2011. [Online; accessed 17-May-2012].

[25] McCoy, Josh and Michael Mateas. “An integrated agent for playing real-time strategy games”. *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, 1313–1318. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL www.aaai.org/Papers/AAAI/2008/AAAI08-208.pdf.

[26] Numpy developers. “Scientific Computing Tools For Python — Numpy”, 2011. URL http://numpy.scipy.org. [Online; accessed 24-Oct-2011].

[27] Obradovic, Darko and Armin Stahl. “Learning by Observing: Case-Based Decision Making in Complex Strategy Games”. *Proceedings of the 31th Annual German Conference on Artificial Intelligence*. Springer, 9 2008.

[28] Ponsen, Marc J. V., Héctor Muñoz Avila, Pieter Spronck, and David W. Aha. “Automatically acquiring domain knowledge for adaptive game AI using evolutionary learning”. *Proceedings of the 17th conference on Innovative applications of artificial intelligence - Volume 3*, IAAI’05, 1535–1540. AAAI Press, 2005. ISBN 1-57735-236-x.

[29] Ponsen, Marc J. V., Héctor Muñoz-Avila, Pieter Spronck, and David W. Aha. “Automatically Generating Game Tactics through Evolutionary Learning”. *AI Magazine*, 27(3):75–84, 2006. URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/1894.

[30] Ponsen, Marc J. V. and Pieter Spronck. “Improving Adaptive Game AI with Evolutionary Learning”. *Proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE-04)*, 389–396. University of Wolverhampton, England, 2004.

[31] Python Developers. “Generators — PythonInfo Wiki”, 2011. URL http://wiki.python.org/moin/Generators. [Online; accessed 24-Oct-2011].

[32] Python Software Foundation. “15.17. ctypes A foreign function library for Python — Python v2.7.3 documentation”, 2011. URL http://docs.python.org/library/ctypes.html. [Online; accessed 24-Oct-2011].

[33] Rosen, Kenneth H. *Discrete Mathematics And Its Applications*. Discrete Mathematics and Its Applications. McGraw-Hill Higher Education, sixth edition, 2006. ISBN 9780073229720.

[34] Russell, S.J. and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, third edition, 2010. ISBN 9780136042594.

[35] Schadd, Frederik, Sander Bakkes, and Pieter Spronck. “Opponent modeling in real-time strategy games”. *Games and Simulation GAMEON*, 3(71):61–68, 2007. URL http://ticc.uvt.nl/~pspronck/pubs/SchaddBakkesSpronckGAMEON07.pdf.

[36] scikit-learn developers. “scikit-learn: machine learning in Python”, 2011. URL http://scikit-learn.org. [Online; accessed 24-Oct-2011].

[37] Spronck, Pieter, Marc Ponsen, and Eric Postma. “Adaptive game AI with dynamic scripting”. *Machine Learning*, 217–248. Kluwer, 2006.

[38] Sutton, Richard S. and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998. ISBN 0262193981.

[39] Synnaeve, Gabriel and Pierre Bessière. “A Bayesian model for opening prediction in RTS games with application to StarCraft”. *Computational Intelligence and Games (CIG)*, 281–288. 2011.

[40] Synnaeve, Gabriel and Pierre Bessière. “A Bayesian Model for Plan Recognition in RTS Games Applied to StarCraft”. *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2011.

[41] Synnaeve, Gabriel and Pierre Bessière. “A Bayesian model for RTS units control applied to StarCraft”. *Computational Intelligence and Games (CIG)*, 190–196. 2011.

[42] The Spring Community. “Spring Engine”, 2011. URL http://www.springrts.com. [Online; accessed 24-Oct-2011].

[43] The Spring Community. “Spring Engine Source Code (develop branch) — Github”, 2011. URL https://github.com/spring/spring. [Computer Program Source Code; Online; accessed 24-Oct-2011].

[44] The Spring Community. “Spring Engine: Strategy and Tactics”, 2011. URL http://springrts.com/wiki/Strategy_and_Tactics. [Online; accessed 24-Oct-2011].

[45] Van Der Heijden, Marcel, Sander Bakkes, and Pieter Spronck. “Dynamic formations in real-time strategy games”. *2008 IEEE Symposium On Computational Intelligence and Games*, 47–54, 2008.

[46] van Rossum, Guido. “Python Programming Language — Official Website”, 2011. URL http://www.python.org. [Online; accessed 24-Oct-2011].

[47] Weber, Ben G. and Michael Mateas. “Case-Based Reasoning for Build Order in Real-Time Strategy Games”. *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 10/2009 2009.

[48] Weber, Ben G. and Michael Mateas. “Conceptual Neighborhoods for Retrieval in Case-Based Reasoning”. *ICCBR ’09: Proceedings of the 8th International Conference on Case-Based Reasoning*, 343–357. Springer-Verlag, Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-02997-4.

[49] Weijers, Stephan. *Real-Time Strategy High-level Planning*. Technical report, University of Twente — Study Tour Pixel 2010, 2010.

[50] Weissgerber, Kurt. *Developing an Effective and Efficient Real Time Strategy Agent for Use as a Computer Generated Force*. Master's thesis, Air Force Institute of Technology, 2010.

[51] Weissgerber, Kurt, Gary B. Lamont, Brett J. Borghetti, and Gilbert L. Peterson. "Determining Solution Space Characteristics for Real-Time Strategy Games and Characterizing Winning Strategies". *International Journal of Computer Games Technology*, 2011.

[52] Wikipedia. "Deep Blue (chess computer) — Wikipedia, the Free Encyclopedia", 2011. URL http://en.wikipedia.org/wiki/Deep_Blue_%28chess_computer%29. [Online; accessed 24-Oct-2011].

[53] Wikipedia. "History of chess — Wikipedia, the Free Encyclopedia", 2011. URL http://en.wikipedia.org/wiki/History_of_chess. [Online; accessed 24-Oct-2011].

[54] Wikipedia. "Real-time strategy — Wikipedia, the Free Encyclopedia", 2011. URL http://en.wikipedia.org/wiki/Real-time_strategy. [Online; accessed 24-Oct-2011].

[55] Wikipedia. "Watson (computer) — Wikipedia, the Free Encyclopedia", 2011. URL http://en.wikipedia.org/wiki/Watson_%28computer%29. [Online; accessed 24-Oct-2011].

[56] Wikipedia. "Military simulation — Wikipedia, the Free Encyclopedia", 2012. URL http://en.wikipedia.org/wiki/Military_simulation. [Online; accessed 17-May-2012].

Vita

Lyall Jonathan Di Trapani completed a B.S. and an M.Eng. in electrical engineering at the University of Louisville in 2006 and 2007, respectively. Both degrees were awarded with the highest honors. During his time at the University of Louisville, he earned the Air Force ROTC Distinguished Graduate Award and the Samuel T. Fife Outstanding Master of Engineering Graduate in Electrical Engineering Award. He is currently pursuing an M.S. in computer science from the Air Force Institute of Technology. He has served in the US Air Force as a communications engineer from 2007 to 2010. He is currently serving as a cyberspace officer at the Air Force Institute of Technology.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
14 June 2012	**Master's Thesis**	Aug 2010 – 14 Jun 2012

4. TITLE AND SUBTITLE
A Real-time Strategy Agent Framework and Strategy Classifier for Computer Generated Forces

5a. CONTRACT NUMBER
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Di Trapani, Lyall J, Capt

5d. PROJECT NUMBER
5e. TASK NUMBER
5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/12-04

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)
11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution A. Approved for Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States

14. ABSTRACT

This research effort is concerned with the advancement of computer generated forces AI for Department of Defense (DoD) military training and education. The vision of this work is agents capable of perceiving and intelligently responding to opponent strategies in real-time. Our research goal is to lay the foundations for such an agent. Six research objectives are defined: 1) Formulate a strategy definition schema effective in defining a range of RTS strategies. 2) Create eight strategy definitions via the schema. 3) Design a real-time agent framework that plays the game according to the given strategy definition. 4) Generate an RTS data set. 5) Create an accurate and fast executing strategy classifier. 6) Find the best counter-strategies for each strategy definition.

The agent framework is used to play the eight strategies against each other and generate a data set of game observations. To classify the data, we first perform feature reduction using principal component analysis or linear discriminant analysis. Two classifier techniques are employed, k-means clustering with k-nearest neighbor and support vector machine. The resulting classifier is 94.1% accurate with an average classification execution speed of 7.14 us. Our research effort has successfully laid the foundations for a dynamic strategy agent.

15. SUBJECT TERMS
Real-time strategy, agent framework, strategy classification, Spring Engine

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
REPORT	**ABSTRACT**	**c. THIS PAGE**			**Gary B. Lamont, Phd**
UU	**UU**	**UU**	**UU**	154	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 x4718 gary.lamont@afit.edu